Delphi Prism e ASP.NET

Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 3

Clube Delphi

Quick Update

Utilizando o DBCtrlGrid

Strings

O que mudou no Delphi 2010

Ano 9 :: Edição 120 :: R$ 11,90

DevMedia group

Migrando para o Delphi 2010

Brinde

Comprando esta edição ganhe 4 vídeo-aulas:

- dbExpress 4 – Criando tabelas via DataProvider
- Delphi Prism e ASP.NET – Master Detail com GridView
- StarTeam – Integração com o MS Project
- StarTeam – Topics

PHP e Boas Práticas

Crie uma classe de acesso a dados usando Design Patterns

Jedi VCL

Seleção dos melhores componentes para Delphi

ISSN 1517990-7

9 771517 990009 00120

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Sumário

Quick Update

06 – Utilizando o DBCtrlGrid
[*Daniel Sobrinho Laporte*]

Delphi

Mão na Massa

08 – Jedi VCL
Aprenda a utilizar os componentes da mais famosa suíte Open-Source
[*Daniel Sobrinho Laporte*]

Expert / Novidades

16 – Strings
O que mudou no Delphi 2010
[*Vitor Luiz Rubio*]

Boas Práticas

Boas Práticas / Tutorial/ Novidades

28 – Migração para Delphi 2010
Aplicações distribuídas fazendo uso da nova arquitetura DataSnap
[*Fabrício Hissao Kawata*]

Easy Delphi

Easy Delphi/ Mini-Curso/ Tutorial/ Web/ Novidades

38 – Delphi Prism e ASP.NET
Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 3
[*Guinther Pauli*]

PHP

PHP/ Web/ Boas Práticas/ Tutorial

45 – Design Patterns
Centralizando o acesso a banco de dados usando o padrão Singleton
[*Dheyson Wildny*]

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Brinde na web desta edição - 4 vídeos! Confira:

4 Vídeos

- **DBX4: criando tabela via DataProvider**
  http://www.devmedia.com.br/post-17036-DBX4--criando-tabela-via-DataProvider-Curso-Novidades-no-Delphi-2010-Parte-20--Final.html

- **MasterDetails: GridView com TemplateFields (Delphi Prism)**
  http://www.devmedia.com.br/post-16919-MasterDetails--GridView-com-TemplateFields-Curso-Delphi-Prism-Parte-44.html

- **Curso StarTeam: Criando uma lista de discussão para os membros da equipe, utilizando Item Topics**
  http://www.devmedia.com.br/post-16502-StarTeam-Curso-basico-Parte-XIV-Organizando-Listas-de-Discussao-para-os-membros-da-equipe-com-o-item-Topics.html

- **Curso StarTeam: Integração com o MS Project**
  http://www.devmedia.com.br/post-16851--Integracao-com-o-MS-Project-Curso-Borland-StarTeam-e-Gerencia-de-Configuracao-Parte-16--Final.html

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.CD** *Senha:* **IOS4**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.CD** Senha: **IOS4**

Clube Delphi

Ano 9 - 120ª Edição 2010 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.laporte@gmail.com

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br

Equipe Editorial
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Capa e Diagramação
Romulo Araujo
romulo@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Cristiany Queiroz – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

EDITORIAL

O Delphi 7 foi durante muito tempo considerado o melhor Delphi de todos os tempos, tanto que ainda hoje faz parte do cotidiano de muitos desenvolvedores que não migraram, por uma série de razões, para versões mais atuais do produto (Delphi 8, 2005, 2006, 2007, 2009). Em uma analogia com o sistema operacional mais famoso do mundo, poderíamos dizer que o Delphi 7 foi o "Windows XP" dos Windows. Essa história mudou com a chegada do Delphi 2010.

Eu poderia citar aqui várias razões pelas quais constatamos que o Delphi 2010 é o sucessor do até Delphi "Highlander". Acredito que a principal é que o IDE não tem mais suporte ao .NET (desde o 2009). O desenvolvimento com .NET está agora centralizado no IDE do Visual Studio, através do Delphi Prism, um IDE robusto para o desenvolvimento de aplicações para essa plataforma. Além disso, temos várias melhorias no ambiente de desenvolvimento que, se acumularmos seus novos recursos desde o Delphi 8, tornam a migração muito atraente. Na VCL, temos vários novos componentes, muitos dos quais já abordamos aqui na ClubeDelphi. DBExpress e DataSnap com várias melhorias, RTTI remodelada, suporte a Multitouch e Gestures, e muitos excelentes recursos na linguagem Delphi Win32, que a tornaram mais parecida com linguagens OO modernas e atuais como C# e Delphi Prism. A migração para Delphi 2010 é natural, e também destaque desta edição.

Na ClubeDelphi edição 70 publiquei uma matéria que sem dúvida foi uma das que mais recebemos feedback por aqui na redação. Tratava do desenvolvimento de aplicações distribuídas com DataSnap usando SOAP e Web Services, de forma que você poderia distribuir clientes Delphi Win32 pela Web, por exemplo em filiais, integrando com um servidor de aplicação centralizado. Os clientes poderiam sair da intranet e serem colocados em qualquer lugar da internet. Uma grande arquitetura e uma excelente alternativa de baixo custo para quem precisa distribuir dados na Web sem adotar uma outra tecnologia. Nesta edição, o Fabrício mostra como migrar esta solução para o Delphi 2010, usando o novo DataSnap. Por outro lado, nosso principal tipo de dado – as strings – sofreu várias alterações, tanto que temos um artigo completo sobre as novidades e mudanças, bem indicadas pelo Vitor Rubio. E prepare-se, Delphi 2011 vem aí e nós já estamos ligados para trazermos todas as novidades em primeira mão aqui na Revista ClubeDelphi!

Ainda nesta edição, o Daniel Laporte traz exemplos com essa que sem dúvida alguma é a mais completa suíte de componentes para Delphi, a Jedi VCL. Na terceira parte da série que estou trazendo sobre o Delphi Prism você vai aprender como é simples e rápido criar aplicações Web ASP.NET que acessam banco de dados (e como isso é parecido com o Delphi Win32). Na sessão PHP, boas práticas no acesso a dados, aprenda a criar uma classe Singleton no artigo do Dheyson.

Boa leitura, sucesso com o Delphi 2010!

Guinther Pauli

guinther@devmedia.com.br

Certificado Delphi 3,5,6,7,2005,2006,Web,Kylix

Microsoft Certified: MCP,MCAD,MCSD.NET,MCTS,MCPD (ASP.NET,C#)

guintherpauli.blogspot.com - twitter.com/guintherpauli

ClubeDelphi PLUS

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

DevMedia group

Perguntas e Respostas

Persistência de objetos

Pergunta respondidas por:
Paulo Quicoli

O Marcos, nosso leitor, precisa de uma dica interessante. Sua equipe de desenvolvimento projetou toda a regra de negócio de seu sistema em classes, aplicando a orientação a objetos. Eles utilizaram classes PODO (Plain Old Delphi Objects), classes que não implementam persistência. Algo como pode ser visto na **Listagem 1**.

Listagem 1. Exemplo de classe PODO

```
TCliente = class
private
  FRG: string;
  FDataNascimento: TDateTime;
  FNome: string;
  procedure SetDataNascimento(const Value: TDateTime);
  procedure SetNome(const Value: string);
  procedure SetRG(const Value: string);
public
   property Nome: string read FNome write SetNome;
   property RG: string read FRG write SetRG;
   property DataNascimento: TDateTime
         read FDataNascimento write SetDataNascimento;
end;
```

Sua dúvida agora é, existe algum framework que salve esses objetos e se encarregue da criação do banco de dados e, além disso, permita utilização dos controles Data-Aware?

Vamos às respostas. Até o momento não existe framework de persistência para Delphi Win32 que seja capaz de persistir objetos PODO. Sendo assim é necessário que suas classes de negócio implementem alguma classe base do framework escolhido. Para persistência posso sugerir os seguintes gratuitos:

tiOPF: Framework muito completo e bem flexível. Desenvolvido por uma empresa australiana, é muito utilizado naquela região. Altamente maduro e confiável, pois está em produção desde o Delphi 5. Oferece seus próprios controles visuais, que agilizam o desenvolvimento de interfaces com componentes especializados, que tratam desde a validação de um objeto até a visualizações complexas como ListView que atua como um Grid, pronto para manutenção de objetos. Permite que os objetos sejam salvos em diversos bancos relacionais e até mesmo em arquivos XML. Também oferece mecanismos para a criação do banco de dados via código. Com ele também é possível desenvolver aplicativos três camadas, onde objetos irão trafegar entre servidor e cliente. A exigência aqui é que suas classes de negócio herdem de TtiObject, classe base do framework. E claro, você poderá utilizar seus controles data-aware mesmo utilizando o tiOPF pois ele disponibiliza um Dataset especial, o TTiDataset que torna os objetos plugáveis nas interfaces DataSet. A versão 3 oferece suporte ao Delphi 2010. Na edição 106 você pode acompanhar o desenvolvimento completo de um sistema utilizando os recursos básicos do tiOPF, incluindo até mesmo listagens. Você pode efetuar seu download em *www.tiopf.com*. Um ponto desfavorável do tiOPF é que ele demanda bastante codificação para utilização dos recursos mais avançados.

Instant Objects (IO): Em recursos é bem similar ao tiOPF, contudo, merece destaque um recurso que permite fazer a atualização do esquema do banco de dados baseando-se no modelo de objetos. Outro ponto importante a ressaltar é que existe um designer para facilitar a criação de classe. Em relação ao tiOPF o IO demanda menos código para operações complexas como por exemplo gerenciamento de relacionamentos e atualização de banco de dados. O ponto desfavorável aqui é a falta de atualização do projeto, que ainda não oferece suporte ao Delphi 2010.

Envie suas dúvidas para nossa equipe de especialistas:

guinther@devmedia.com.br
pauloquicoli@devmedia.com.br
daniel.sobrinho@devmedia.com.br
rodrigo.carreiro@devmedia.com.br

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!
Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Quick Update:
Utilizando o DBCtrlGrid

Daniel Sobrinho Laporte
daniel.sobrinho@devmedia.com.br
daniel.laporte@gmail.com
www.twitter.com/daniel_laporte
Colunista e editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

A grande maioria dos desenvolvedores trabalha com campos Data-Aware como DBNavigator, DBGrid, DBEdit, DBText entre muitos outros. Muitas vezes utilizamos estes componentes em situações diversas para realizar as mais diferentes tarefas possíveis e acabamos deixando de explorar alguns componentes com comportamentos/características muito interessantes, que com certeza podem nos auxiliar em inúmeras situações. Se você não conhece a DBCtrlGrid, vai apreciar o que este componente pode fazer.

Embora seja pouco visto em aplicações, pouco explorado e mal aproveitado, este componente possui um comportamento de "Container", onde tudo que esteja dentro de sua área útil, será reproduzido na próxima inserção, deste modo você pode exibir imagens e muitas outras coisas que sua criatividade permitir. Em nosso exemplo criaremos uma aplicação que simule a entrada de uma "grade de calçados" de um software qualquer para ilustrar sua funcionalidade.

Abra seu RAD Studio e crie uma nova aplicação. Insira no formulário um ClientDataSet renomeando-o para cdsMemoria, um DataSource renomeado para dsrMemoria, um DBCtrlGrid e um DBNavigator. Ligue o ClientDataSet ao DataSource por meio da propriedade DataSet, em seguida conecte o DataSource ao DBNavigator e à DBCtrlGrid também através da propriedade DataSet. Observe o Layout de exemplo na **Figura 1.**

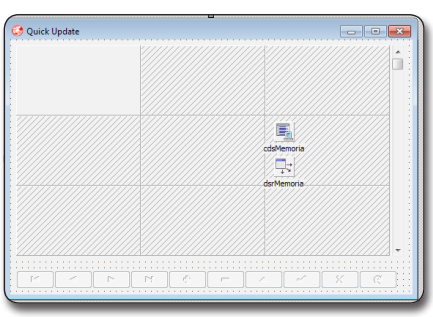


Figura 1. Layout do formulário

Veja na **Figura 1** que temos dentro do componente uma área cinza que não está marcada pelas listras diagonais. Esta área é justamente a que é especificada para a reprodução, assim sendo faremos com que a grade seja inserida e visualizada dentro do componente de forma a ficar automaticamente ordenada, para isso criaremos dois campos em nosso ClientDataSet, que funcionará como tabela de memória. Dê um duplo clique sobre o cdsMemoria e crie dois campos de tipo Integer, o primeiro deverá se chamar Numero e o segundo Quantidade. Feito isso realizaremos a ordenação da grade de acordo com a numeração, portanto, informe à propriedade IndexFieldNames o campo Numero para que tenhamos finalizado as alterações do ClientDataSet.

Para a digitação dos valores faremos com que o usuário informe os campos através de DBEdits. Insira dentro da área cinza em destaque dois Labels com os Captions de Numeração e Quantidade. Adicione também dois DBTexts e ligue-os ao dsrMemoria, informando nos DataFields respectivamente os campos Numero e Quantidade. Para finalizar, no evento OnCreate do formulário faça a codificação necessária para iniciar o desenvolvimento com a tabela de memória:

```
cdsMemoria.Close;
cdsMemoria.CreateDataSet;
```

Como nosso cdsMemoria já estará ordenado, na própria inserção a Grid será automaticamente re-ordenada a cada registro. Observe que os DBEdits e os Labels são automaticamente reproduzidos no momento de uma nova inserção, dispensando a necessidade de criá-los em tempo de execução por exemplo, ou de criar uma grade com quantidade e numeração limitadas, fazendo com que a definição da nossa grade seja ilimitada. Veja o resultado final na **Figura 2.** ●

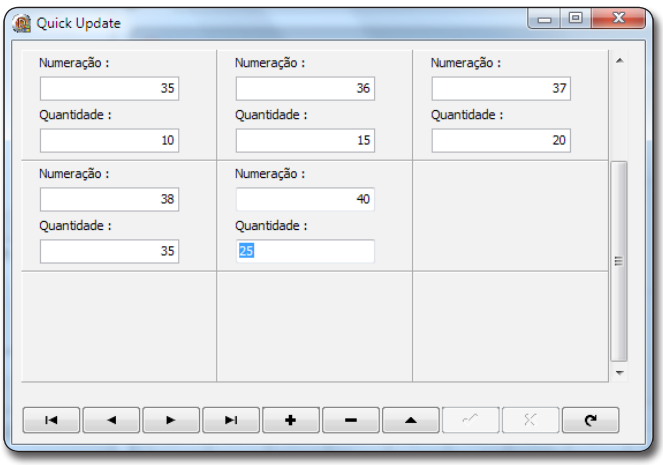


Figura 2. DBCtrlGrid em funcionamento

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Existem coisas que não conseguimos ficar sem!

...só pra lembrar, sua assinatura pode estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

Para mais informações:
www.devmedia.com.br/central

DevMedia GROUP

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Jedi VCL

Aprenda a utilizar os componentes da mais famosa suíte Open-Source

Daniel Sobrinho Laporte

daniel.sobrinho@devmedia.com.br
daniel.laporte@gmail.com
www.twitter.com/daniel_laporte

Colunista e editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

Resumo do DevMan

Embora tenhamos uma grande quantidade de componentes disponíveis no Delphi, sempre nos surge alguma ideia sobre determinada funcionalidade ou componente que gostaríamos de ter no IDE. Se você é fã de componentes de terceiros e ainda não conhece a biblioteca Jedi VCL, eis aqui uma ótima oportunidade de conhecer um pouco sobre todo o poder de uma das melhores, senão a melhor coleção de componentes Open-Source para Delphi.

Componentes de terceiros sempre são tratados como um assunto polêmico entre os desenvolvedores. Isto porque muitos são contra a dependência que os componentes podem causar em uma aplicação, tornando a continuidade ou evolução do projeto

Do que trata o artigo

Apresentar a biblioteca Jedi VCL, abordando uma seleção de componentes da suíte. Conheceremos componentes que auxiliam o desenvolvimento de interfaces bem elaboradas, validação, entre mais. Tudo isso em um passo-a-passo que aborda desde instalação à utilização.

Para que serve

Agregar funcionalidades em nossas aplicações e tornar o desenvolvimento mais simples. Devido à grande quantidade de componentes que a suíte possui você encontrará praticamente tudo que necessita para lhe facilitar o desenvolvimento, dispensando a necessidade de criação de componentes ou mesmo funcionalidades que possivelmente já existem.

Em que situação o tema é útil

No Delphi possuímos uma grande gama de componentes de terceiros, porém muitas delas são pagas, o que às vezes os desenvolvedores evitam. A Jedi VCL é uma biblioteca Open-Source que possui contribuições constantes, sendo muitas vezes premiada como uma das melhores suítes de componentes para Delphi. Desta forma é possível agregarmos funcionalidades ao nosso ambiente sem custo algum.

um pouco mais difícil. Outros alegam que a adição de uma grande quantidade de componentes ao IDE pode causar uma lentidão na inicialização, o que de fato não deixa de ser uma verdade, ou mesmo tornar o executável do seu projeto maior. A verdade é que embora tenhamos todos estes argumentos, o mercado está cheio de suítes, como a famosa DevExpress e a InfoPower por exemplo, ambas pagas.

Para quem possui certa experiência muitas vezes a opção é desenvolver os próprios componentes, para quem não tem, basta uma breve pesquisa na Web para encontrar um componente que adéque à sua necessidade.

Quem geralmente é habituado a trabalhar com componentes de terceiros, e desenvolve com o Delphi há algum tempo, certamente já ouviu falar da suíte RxLib. A RxLib é uma biblioteca que acompanha a evolução do Delphi, e que ainda hoje possui sua versão disponível para o Delphi 2010. Muitos desenvolvedores ainda a utilizam para manter a compatibilidade de seus projetos. Você pode estar se perguntando por que comentar da RxLib se estamos abordando a Jedi VCL. Simplesmente porque a Jedi VCL agora inclui a RxLib.

A quantidade de componentes da biblioteca é muito grande, e selecionar componentes para descrever aqui realmente é uma tarefa difícil, porém tentarei dividir em alguns tópicos para que seja possível conhecermos um pouco sobre as principais funcionalidades que podem ser utilizadas em seu desenvolvimento cotidiano.

Iniciaremos este artigo abordando a instalação da suíte, que por sinal é muito simples, e no decorrer iremos fazer pequenos projetos para ilustrar a aplicação e funcionalidade dos componentes.

Instalando a suíte

Após ter realizado o download da biblioteca (ver sessão Links ao final do artigo), descompacte o arquivo em uma pasta de sua preferência. Por padrão teremos duas pastas denominadas JVCL e JCL. Primeiramente instalaremos o conteúdo da pasta JCL, portanto abra a pasta e execute o arquivo Install.bat. Você notará um instalador como o da **Figura 1.** Perceba na imagem que se você possui mais de uma versão de Delphi instalada em seu micro, você terá as versões exibidas através de abas, bastando apenas marcar/desmarcar a versão que você não deseja que seja instalada.

Nota:

Seu Delphi deve estar fechado para realizar a instalação da suíte.

Após ter selecionado o que deve ser instalado apenas clique no botão Install e aguarde até que o instalador finalize o processo, terminando, acesse o diretório JVCL e clique no arquivo Install.bat para que o novo instalador seja executado. A **Figura 2** ilustra o novo instalador.

Como você pode perceber na **Figura 2** temos três opções na tela. A primeira é utilizada para realizar uma atualização ou uma nova instalação, a segunda refere-se apenas à compilação

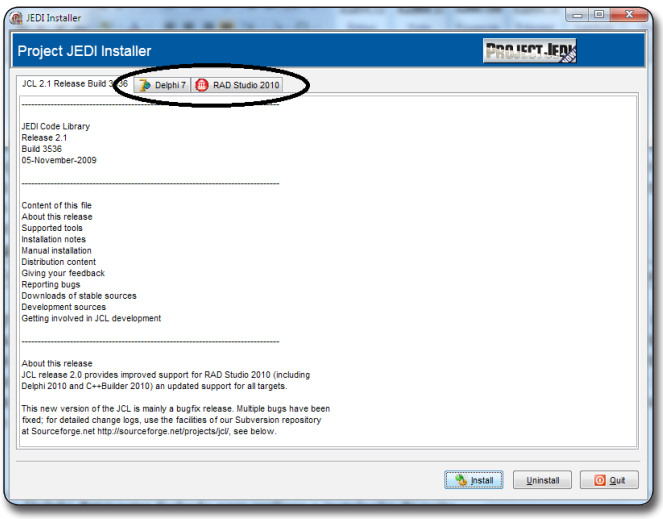


Figura 1. Instalador JCL da suíte Jedi

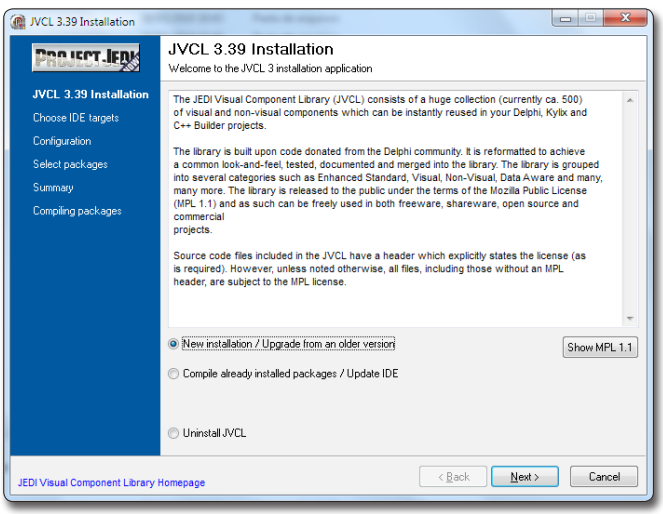


Figura 2. Instalação da JVCL

dos pacotes instalados, realizando apenas um Upgrade destes pacotes no IDE e a terceira e última realiza a desinstalação da suíte. Selecione por padrão a primeira opção e clique em Next. Na próxima tela você selecionará em qual dos IDEs instalados no seu micro você deseja configurar a suíte, novamente dê Next após selecionar. Por último, será exibida uma tela que permite configurar opções de instalação. Por padrão deixaremos como está, clicando em Next e aguardando o final da instalação. Após estes passos estamos prontos para utilizar a suíte.

Nota:

Um detalhe interessante da biblioteca Jedi é que até mesmo o instalador do projeto é desenvolvido em Delphi e como o projeto é Open-Source, o código fonte está inteiramente disponível para seu estudo e alterações.

Personalizando o visual

Primeiramente veremos alguns componentes gráficos que nos proporcionam uma criação de interfaces mais agradáveis e

interessantes. O primeiro componente que veremos é encontrado na paleta Jv Application, Forms. Trata-se do JvTransparentForm. Este componente é perfeito para criar telas de Login, Splash Screens ou qualquer outra janela que você deseja que possua um efeito diferente, basta apenas um pouco de criatividade. Primeiramente abra um programa de tratamento de imagem qualquer, como o Paint, por exemplo, e crie uma imagem do tamanho que desejar. No meu caso criei uma imagem de 350 de altura x 500 de largura. Observe na **Figura 3** o que foi feito.

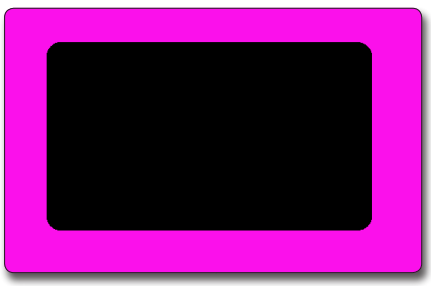


Figura 3. Imagem criada no Paint

Feita a imagem crie um novo projeto no RAD Studio e adicione ao formulário um JvTransparentForm. Observe que o componente possui apenas quatro propriedades: Name, Active, AutoSize e Mask. Setando a propriedade Active para True você notará que o formulário perderá as bordas, o que não nos exibe nenhum efeito, porém, agora entenderemos o porquê da imagem que criamos. Dê um clique sobre a propriedade Mask e no botão Load indique a imagem que criamos no Paint, em seguida, marque novamente a propriedade Active para True e veja o que acontece ao formulário. Assim como quando trabalhamos com ícones em um ImageList por exemplo, a propriedade Mask considera o fundo colorido (rosa), como a área que não faz parte da imagem, sendo assim, o formulário adotará o formato que estiver desenhado no fundo preto. Veja o resultado do nosso exemplo na **Figura 4.**

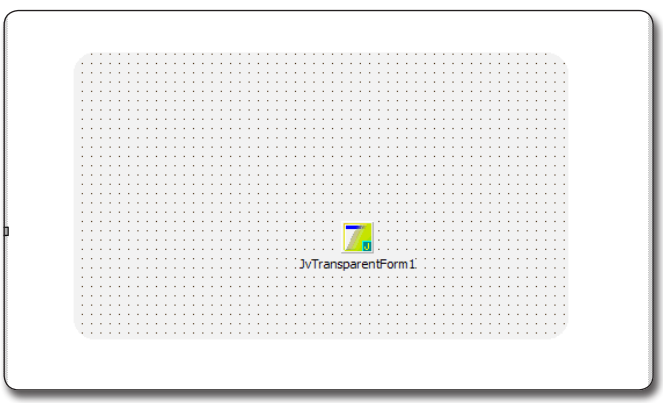


Figura 4. Formato adotado pela propriedade Mask com a imagem

A seguir apresentarei alguns componentes visuais muito interessantes que podem ser utilizados em conjunto para personalizar o formulário principal de sua aplicação por exemplo. Crie um novo projeto no RAD Studio e adicione um Panel, configurando seu Align para alTop e em seguida, na paleta Jv Bars, Panels adicione um JvGradient dentro do Panel. Perceba que como o próprio nome sugere este componente permite um efeito gradiente no formulário. As principais propriedades são StartColor e EndColor, que permitem configurar as cores de início e fim do gradiente, e Style que define seu comportamento ou tipo de gradiente se desejar.

Com um pouco de criatividade podemos utilizar este componente com tantos outros da biblioteca Jedi que permitem com que você realize a personalização de tudo que desejar. Selecione a paleta Jv Buttons e insira dois JvOfficeColorButton ao formulário, adicione também dois Labels com Caption de Cor inicial e Cor final respectivamente, e por último um Button com Caption de Aplicar Cores. Note na **Figura 5** o visual de nosso formulário.

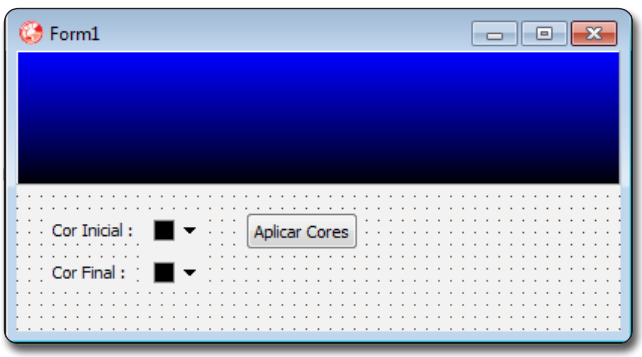


Figura 5. Visual com os componentes JvOfficeColorButton e JvGradient

No evento OnClick do botão inserido adicione a seguinte codificação:

```
JvGradient1.StartColor := JvOfficeColorButton1.SelectedColor;
JvGradient1.EndColor  := JvOfficeColorButton2.SelectedColor;
```

Como você pode notar a implementação anterior sugere que as cores selecionadas nos botões JvOfficeColorButton sejam atribuídas às cores iniciais e finais do gradiente. Dessa forma você poderia, por exemplo, salvar estes dados em um XML através do ClientDataSet e permitir que cada usuário definisse sua cor de gradiente que mais lhe agrada, tornando sua aplicação mais customizada.

Ainda abordando a personalização de aplicações podemos fazer algo parecido com o menu de nossa aplicação. Selecione na aba Jv Non Visual um JvMainMenu. Repare que no componente JvMainMenu nós possuímos uma propriedade denominada ItemPainter. Com esta propriedade podemos selecionar componentes que possibilitarão realizar um efeito no JvMainMenu, permitindo selecionar detalhes como as cores de sombra quando o cursor estiver posicionado sobre o item ou a cor da lateral de cada um dos itens. Para personalizar o JvMainMenu podemos optar por utilizar cinco componentes diferentes de customização, sendo: JvOfficeMenuItemPainter, JvBtnMenuItemPainter, JvStandardMenuItemPainter, JvOwnerDrawMenuItemPainter ou JvXPMenuItemPainter. Todos estes componentes encontram-se na paleta Jv Non Visual e podem ser configurados através da propriedade ItemPainter,

porém, utilizaremos em nosso exemplo apenas o JvXPMenuItemPainter para demonstração.

Após inserir o componente, ligue à propriedade ItemPainter do JvMainMenu o componente JvxXPMenuItemPainter, em seguida dê um duplo clique sobre o JvMainMenu e insira alguns itens de sua preferência. Você pode inclusive personalizar as cores deste utilizando os mesmos métodos que realizamos a personalização do gradiente com os botões JvOfficeColorButton. Note o resultado final na **Figura 6.**

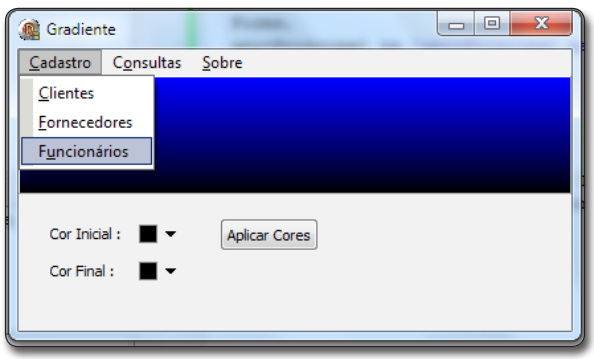


Figura 6. Tela final com o JvMainMenu

Componentes Data-Aware

Tradicionalmente no Delphi utilizamos a paleta Data Controls para nos referirmos aos componentes que se conectam diretamente ao banco de dados. É claro que temos uma quantidade bem limitada de controles deste tipo, mas, que certamente nos atende na maioria das vezes. Conheceremos alguns controles da biblioteca que funcionam do mesmo modo que o convencional, porém, com maiores funcionalidades e outros que são novidades.

Crie um novo projeto no RAD Studio e adicione ao projeto um ClientDataSet e um DataSource nomeando-os para cds e dsr respectivamente. Em seguida dê um duplo clique sobre o ClientDataSet para que seja aberto o Fields Editor. Nele crie os campos conforme ilustra a **Tabela 1.**

Name	Type	Size	Expression
Funcionario	String	250	
Salario	Currency	-	
DataCadastro	Date		
TetoEmpresa	Aggregate		SUM(Salario)
TotalRegistros	Aggregate		COUNT(Funcionario)

Tabela 1. Campos do cds

Após ter criado os campos, ainda no Fields Editor, selecione os campos TetoEmpresa e TotalRegistros e sete a propriedade Active para True. Em seguida, selecione o cds e ative a propriedade AggregatesActive para que os campos agregados comecem a funcionar. Ligue o dsr ao cds através da propriedade DataSource e codifique no evento OnCreate do formulário a seguinte implementação:

```
cds.CreateDataSet;
```

Com isso o ClientDataSet trabalhará como uma tabela de memória, nos permitindo inserir e manipular dados em tempo de execução e sem a necessidade de configurar uma conexão com o banco de dados.

Para iniciar vamos comentar sobre a Grid. Quem trabalha com DBGrid certamente sabe que é um excelente componente de exibição de dados, no entanto, muitas vezes desejamos realizar tarefas simples que exigem modificações no código fonte e re-compilação de nosso projeto. Assim como no Delphi que possuímos a paleta Data Controls, basearemos nossos exemplos na paleta Jv Data Controls. Adicione ao formulário uma JvDBUltimGrid para iniciarmos.

Primeiramente vamos configurá-la como a DBGrid tradicional, para isso indique o dsr na propriedade DataSource e com um duplo clique sobre a Grid, clique com o botão direito do mouse no novo editor e selecione Add All Fields. Configure a disposição e o tamanho dos campos conforme sua preferência. Aproveite e adicione um DBNavigator ao formulário para realizarmos a manipulação de dados. Assim como na Grid, indique o dsr à propriedade DataSource do DBNavigator, feito isso o nosso formulário estará pronto para ser utilizado.

Até aqui nada demais, porém, você como desenvolvedor ao menos uma vez deve, por opção própria ou mesmo por necessidade, ter necessitado zebrar uma Grid. Se você nunca ouviu falar neste termo, zebrar a Grid significa alternar as cores de suas linhas. Com certeza com uma simples busca na Web você encontrará dezenas de códigos para realizar este efeito, mas esta Grid já possui este recurso incorporado bastando apenas configurar uma única propriedade. Selecione a propriedade AlternateRowColor e configure a cor que deseja que seja exibida quando a linha for alternada, para o meu caso utilizei clGray. Execute a aplicação, insira mais de duas linhas e observe o resultado conforme ilustra a **Figura 7**.

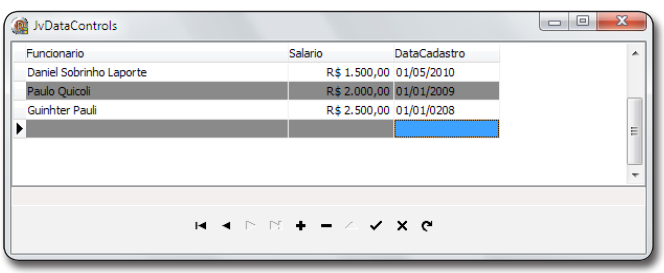


Figura 7. Exemplo de Grid zebrada

Este recurso de zebrar a Grid é muito simples e dispensa alterações de código-fonte ou re-compilação do projeto. Outra opção que veremos agora é a seleção de colunas.

Muitas vezes, fazemos uma tela qualquer para o usuário e este nos solicita que seja adicionada uma coluna qualquer, às vezes esta coluna já está presente em nossa seleção de dados, já está nos Fields do DataSet mas achamos desnecessário e não incluímos este campo na visualização da DBGrid. Caso o usuário necessite deste determinado campo, teremos que abrir o projeto, adicionar o campo na DBGrid, compilar o projeto e distribuí-lo para o cliente. Habilite a propriedade TitleArrow

e note que foi inserida uma seta apontada pra baixo, na própria Grid ao lado do primeiro campo. Novamente execute a aplicação e clique na seta que apareceu você terá um resultado semelhante ao da **Figura 8.**

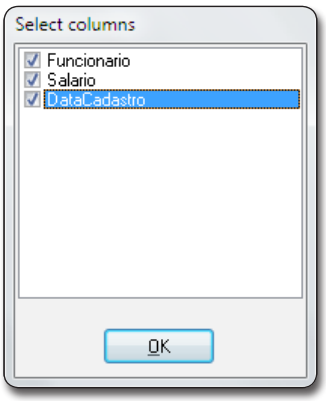


Figura 8. Janela de seleção de colunas

Observando a **Figura 8** você notará que o título da janela está em inglês, caso você desmarque as três colunas do nosso exemplo também receberá uma mensagem de aviso em inglês. Para customizar ainda mais nossa aplicação, há a possibilidade de configurarmos estas mensagens. Basta expandir a propriedade JvSelectDialogColumnStrings e customizar o Caption (mensagem do topo da janela), NoSelectionWarning (mensagem de aviso que será exibida ao desmarcar todos os campos) e por último OK (botão).

De fato temos vários componentes comuns, não Data-Aware, na qual gostaríamos que se tornassem Data-Aware. Se pararmos para analisar talvez nem seja uma tarefa trabalhosa portá-los, porém, se você já utiliza a biblioteca Jedi não há motivos para não usá-los. Um exemplo disto é o componente JvDBSpinEdit. Adicione um ao formulário, juntamente com um Label. Configure o Caption do Label para "Salário", ligue ao JvdBSpinEdit o dsr na propriedade DataSource e indique o DataField Salario.

O interessante deste componente é que nos possibilita trabalhar com um range (intervalo) de valores que é automaticamente validado pelo próprio componente. Para fazer um teste ative as propriedades CheckMaxValue e CheckMinValue em seguida informe os valores 1 para MinValue e 1500 para MaxValue. Execute a aplicação e veja que você não conseguirá inserir um salário fora deste intervalo, pois o componente automaticamente se encarrega de validar o intervalo informado.

Outra característica muito útil é que você pode validar o tipo de campo através da propriedade ValueType podendo ser: vtInteger para valores inteiros, vtFloat para números reais e vtHex para valores hexadecimais. Ainda podemos adequar ainda mais à nossa necessidade, especificando o ValueType para vtFloat e configurando o número de casas decimais na propriedade Decimal.

Se você alguma vez quis ter um DateTimePicker que gravasse dados, no Jedi você encontra este componente. Adicione o componente JvDBDateTimePicker e assim como os outros componentes Data-Aware, informe o dsr ao DataSource e o campo DataCadastro no DataField. Execute a aplicação e veja que ao selecionar uma data, automaticamente a Grid é preenchida com a data selecionada. Assim como o DateTimePicker tradicional, você pode selecionar a data diretamente no calendário, clicando no ícone do lado da data. Observe a **Figura 9.**

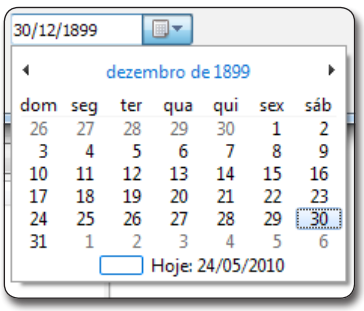


Figura 9. JVDBDateTimePicker em ação

Você deve estar se perguntando por que criamos dois campos do tipo Aggregate se até agora nada fizemos com eles. Pois bem, a razão para a criação destes dois campos é que vamos apresentar o último componente Data-Aware deste artigo e utilizaremos estes campos agregados. Apresentaremos aqui o JvDBGridFooter, que nada mais é do que um rodapé para a JvDBGrid ou JvDBUltimGrid. Adicione um ao formulário do projeto e mude seu Align para alTop. Ligue à propriedade DBGrid ao nosso Grid, perceba que automaticamente ele preencherá a propriedade DataSource com o dsr que está ligado ao Grid. O próximo passo é a criação dos rodapés que queremos que sejam exibidos. Clique na propriedade TFooterColums e note que será exibido um editor que permite inserir colunas. Insira duas colunas, preenchendo suas propriedades DisplayMask e FieldName para Funcionario e Salario respectivamente. Um fator interessante é que queremos totalizar a quantidade de registros e o salário total dos funcionários cadastrados, e informamos em FieldName: Funcionario e Salario. Simples. Isto foi feito para que a Footer entenda que o posicionamento dos campos que queremos exibir seja relativo à posição dos campos de mesmo nome na Grid, ou seja, teremos um rodapé abaixo da coluna Funcionario e outro abaixo da coluna Salario. Feito isso, precisamos manipular o evento que fará a exibição do conteúdo na Footer, para isso, dê um duplo clique no evento OnCalculate do JvDBGridFooter e faça a implementação da **Listagem 1.**

O código da **Listagem 1** é muito simples como podemos observar. Primeiramente verificamos se o cds está ativo, para então realizar os cálculos. Na Linha 05 verificamos se o parâmetro FieldName (que indica a coluna que está sendo passada no momento), equivale à Funcionario, caso seja verdadeira a condição, na Linha 07 verificamos se o campo TotalRegistros não está nulo e atribuímos seu valor concatenado da palavra "Salário total:" à variável CalcValue, responsável por exibir o valor na Footer, caso o campo esteja nulo a Linha 10 fará com que a variável receba o conteúdo em branco. O mesmo ocorre entre as Linhas 11 e 20, porém validando a coluna

Listagem 1. Implementação do evento OnCalculate do JvDBGridFooter

```
01 procedure TForm1.JvDBGridFooter1Calculate
     (Sender: TJvDBGridFooter;
     const FieldName: string; var CalcValue: Variant);
02 begin
03   if cds.Active then
04   begin
05     if AnsiSameText(FieldName, 'Funcionario') then
06     begin
07       if not VarIsNull(cdsTotalRegistros.Value) then
08         CalcValue := 'Quantidade de registros : ' +
           VarToStr(cdsTotalRegistros.Value)
09       else
10         CalcValue := '';
11     end
12     else if AnsiSameText(FieldName, 'Salario') then
13     begin
14       if not VarIsNull(cdsTetoEmpresa.Value) then
15         CalcValue := 'Salário Total : ' + VarToStr
           (cdsTetoEmpresa.Value)
16       else
17         CalcValue := '';
18     end;
19   end;
20 end;
```

Salario e o campo TetoEmpresa, já que este evento ocorrerá para todas as colunas presentes na Grid. Um detalhe interessante deste componente é que você pode redimensionar as colunas da Grid que automaticamente o Footer será redimensionado também. O resultado final você pode observar na **Figura 10**.

Com isso finalizamos a parte que aborda alguns dos componentes Data-Aware existentes na biblioteca. A seguir o que veremos são alguns componentes diversos que podem agregar funcionalidades interessantes em nossa aplicação, ou mesmo facilitar nosso desenvolvimento cotidiano.

Validações

Em se tratando de validações ou captura de erros, podemos utilizar em nossas aplicações o componente ApplicationEvents, o qual recebe as exceções através do parâmetro "E" contido em sua assinatura. Na biblioteca Jedi VCL podemos utilizar um conjunto de componentes que podem dar um efeito diferenciado em nossa aplicação, realizando tarefas de validação de componentes e nos poupando algumas linhas de código. Falaremos um pouco sobre os componentes JvValidators, JvErrorIndicator e JvValidationSummary da aba JvValidators.

Para iniciar, crie um novo projeto e adicione ao formulário os três componentes citados, dois Labels, um Button, um ImageList e um componente JvValidatorEdit da aba Jv Edits. Ao terminar a inclusão dos componentes configure sua tela como ilustrado na **Figura 11**.

Primeiramente note o componente JvValidators. Este é responsável por criar as regras de validação

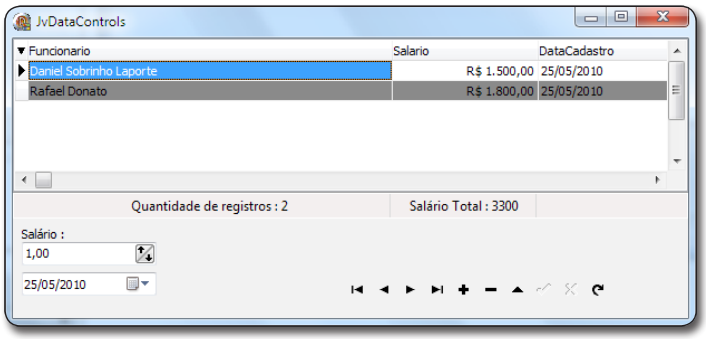


Figura 10. Utilizando o JvDBGridFooter

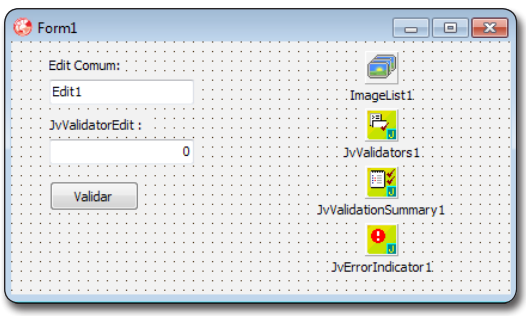
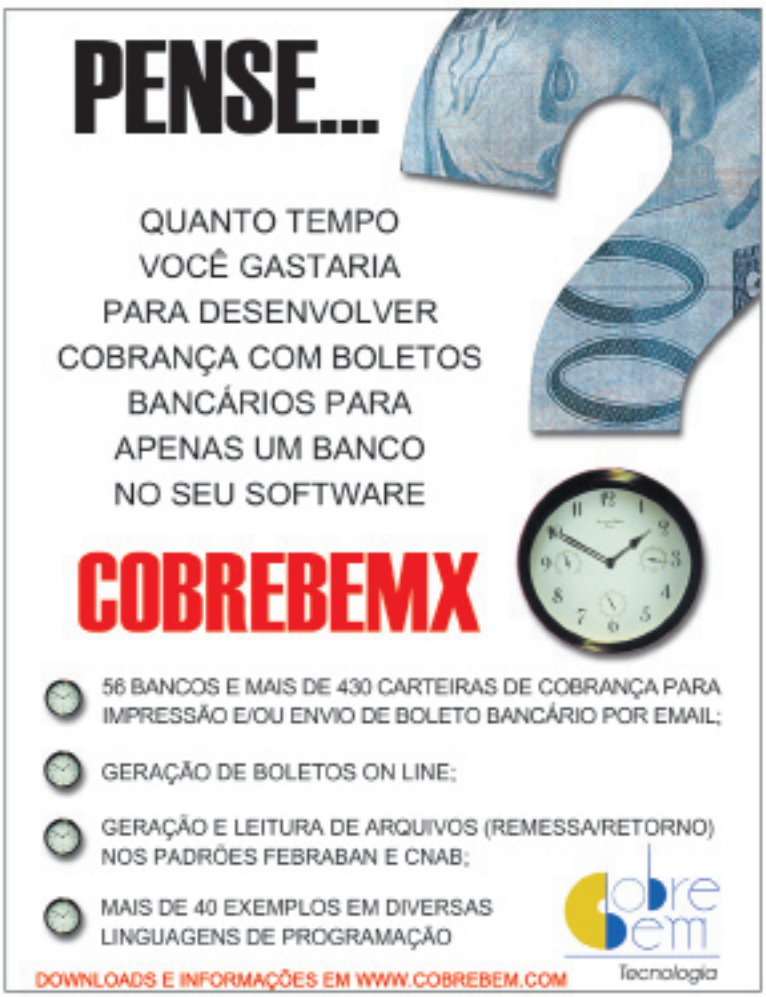


Figura 11. Aplicação com componentes de validação



que você deseja que sua aplicação seja capaz de fazer. Primeiramente clique sobre o componente e note que temos as propriedades ErrorIndicator e ValidationSummary em vermelho, selecione então os respectivos componentes para as propriedades. Para criar uma nova regra de validação simplesmente dê um duplo clique sobre o componente JvValidators. A **Tabela 2** ilustra os tipos de validação que podemos fazer com o componente.

Validação	Funcionalidade
Required Field Validator	Obriga o preenchimento do controle.
Compare Validator	Realiza comparações entre conteúdos dos controles.
Range Validator	Valida o preenchimento entre intervalo de valores.
Regular Expression Validator	Realiza validações através de expressões regulares.
Custom Validator	Permite a criação de funções personalizadas para validação.
Controls Compare Validator	Compara valores de controles diferentes.

Tabela 2. Tipos de validação permitidos para o JvValidators

Para exemplificar vamos realizar um exemplo com o Required FieldValidator e um com o Range Validator. Portanto, ao dar um duplo clique sobre o componente JvValidators acrescente uma validação de cada um dos dois tipos mencionados. Observe a janela na **Figura 12.**

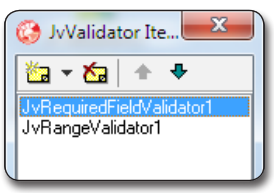


Figura 12. Janela de criação de validação

Com as opções devidamente criadas configuraremos primeiramente o JvRequiredFieldValidator. Note que temos no Object Inspector a opção ControlToValidate que refere-se ao controle que desejamos que seja realizada a validação, selecione o Edit1 para as opções ControlToValidate e ErrorControl. Em ErrorMessage configure a mensagem "Algo deve ser digitado!" e em PropertyToValidate informe a propriedade Text. Perceba que em ControlToValidate você pode selecionar os controles existentes no formulário, e em PropertyToValidate é possível selecionar qualquer propriedade daquele determinado controle escolhido. Feito isso, configure as propriedades do JvRangeValidator. Selecione primeiramente em ControlToValidate e ErrorControl o componente JvValidateEdit, em ErrorMessage informe "Fora do intervalo de valores!" e em PropertyToValidate configure a propriedade Value. Para configurarmos o intervalo de valores temos as propriedades MaximumValue e MinimumValue e expandindo-as podemos informar o seu tipo, na propriedade Type. Informe o valor 1 para MinimumValue e 10 para MaximumValue e ambos configure o Type para Integer.

Com isso terminamos a validação que será realizada nos componentes da tela. Agora que fizemos o fundamental vamos abordar um pouco os demais componentes. O JvValidationSummary é responsável por armazenar e capturar as mensagens configuradas no JvValidators, permitindo por exemplo exibi-las através de diálogos ou mesmo em arquivos de log que podemos criar manualmente. O componente JvErrorIndicator é utilizado para exibir o efeito de erro na frente do controle que será validado. Nele temos a propriedade BlinkStyle que possibilita configurar o tipo de efeito de piscar que será exibido, também temos a possibilidade de configurar um ImageList para exibir ícones de erro de acordo com nossa preferência. Por padrão é exibido um exclamação ( ), no meu caso informei um ícone de erro no ImageList, e configurei a propriedade Images apontando para o ImageList. Por último temos o JvValidatorEdit que possui características muito interessantes e que de fato o tornam muito mais poderoso que um Edit comum. A propriedade DisplayFormat permite configurar o tipo de informação que o componente permitirá que o usuário informe. Lembra que sempre realizávamos validações de intervalos de valores numéricos através do evento OnKeyDown do Edit? Neste caso o próprio componente já trata de realizar este tipo de validação. Deixe configurado nesta propriedade o DisplayFormat dfInteger. A propriedade CheckChars permite que você configure os caracteres que serão checados, deixaremos como padrão os valores numéricos de 0 a 9. Outro fator interessante é que através da propriedade DataConnector nós podemos fazer com que esse Edit trabalhe como um componente Data-Aware, diretamente no banco de dados. Se você deseja realizar validações entre intervalos numéricos pode inclusive utilizar as propriedades MaxValue e MinValue para informar os valores, dispensando assim a necessidade dos demais componentes que exibimos anteriormente, como este não é o nosso caso vamos deixar os valores zerados. Após terminar as devidas configurações do componente, dê um duplo clique sobre o Button e configure o evento OnClick com a seguinte implementação:

```
JvValidators1.Validate;
```

Ao clicar no botão o método Validate do componente JvValidators será acionado, e as respectivas validações configuradas serão executadas. Limpe o conteúdo do Edit e execute a aplicação, em seguida clique no Button. Perceba que você verá um ícone piscando na frente dos controles, e posicionando o ponteiro do mouse sobre os ícones você notará as mensagens que configuramos em forma de Hints. O resultado final é mostrado na **Figura 13**.

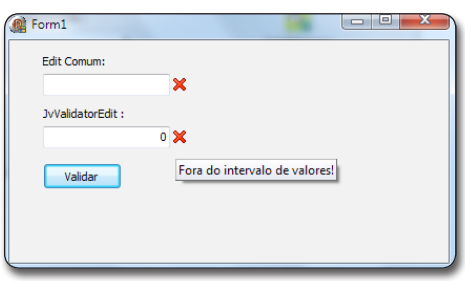


Figura 13. Exemplo final de validação

Conclusão

Como pudemos ver a suíte de componentes Jedi VCL possui uma grande variedade de opções que podem atender a pelo menos uma de suas necessidades. O melhor de tudo isso é que é uma biblioteca inteiramente gratuita, com código-fonte disponível, o que nos permite expandir nossos conhecimentos sobre criação de componentes bem como o conhecimento da própria biblioteca.

Mostramos aqui apenas uma mínima parte do que a biblioteca possui, deixando que você sinta-se à vontade para utilizar o que desejar. Tire proveito desta excelente biblioteca e crie aplicações mais criativas. Um grande abraço e até a próxima! ●

Links

Download biblioteca Jedi VCL
http://sourceforge.net/projects/jvcl/

Blog do autor
http://twitter.com/daniel_laporte

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Strings

O que mudou no Delphi 2010

Do que trata o artigo

Este artigo trata sobre Strings, como funcionam, do que são formadas, as mudanças que houve e como podemos fazer para migrar para o novo Delphi. Contaremos uma breve história sobre as mudanças ocorridas desde o Delphi 3 e daremos alguns exemplos.

Para que serve

Programas feitos em Delphi 2010 que usam bibliotecas/dll's feitas em Delphi 7 não vão funcionar, devido a problemas relativos ao tamanho e organização dos dados. O mesmo se aplica a programas feitos em Delphi 7 utilizando bibliotecas compiladas em Delphi 2010.

Em que situação o tema é útil

O tema é útil tanto para migração de aplicações para a nova versão do Delphi como também para compatibilização com bibliotecas legadas. Também será útil como uma introdução ao Unicode e a primeira etapa para a globalização de sua aplicação.

Resumo do DevMan

Se ao usar uma dll legada com o novo Delphi você recebeu uma mensagem de Access Violation ou até viu algumas Strings suas corromperem-se com caracteres inválidos e ilegíveis isso aconteceu por causa do novo formato de Strings. Vamos explicar detalhadamente o que mudou, o que você pode fazer e até onde você pode chegar com as novas características do Delphi. Além disso, seremos introduzidos ao Unicode e suas possibilidades. Criaremos exemplos com dll's que imitam situações da vida real e solucionaremos alguns dos problemas mais comuns.

Vitor Luiz Rubio
vitorrubio@gmail.com
Analista de Sistemas Sr. na Editora Revista dos Tribunais. Trabalha com Delphi desde a versão 3. Formado em Processamento de Dados pela FATEC-SP.

Strings são um dos maiores avanços na programação. Elas abstraem sequências de caracteres que formam palavras e frases, e nos permitem trabalhar com essas sequências de forma transparente, sem a necessidade de trabalhar com vetores de caracteres.

Infelizmente não há um padrão na implementação de Strings entre as várias linguagens e frameworks. Enquanto no

C++ as strings são ponteiros para vetores de caracteres com tamanho variável e terminadas em null, as Strings no Delphi são ponteiros para estruturas contendo informações sobre comprimento, contagem de referências e o vetor de caracteres propriamente dito.

Já os caracteres sempre foram caracteres, e sua implementação sempre foi padrão, certo? Errado. Ao longo dos anos a definição de caracteres também sofreu mudanças. Algumas linguagens sofreram a mudança cedo, outras sofreram essa mudança mais tarde. Algumas linguagens já nasceram com o conceito de caractere Unicode. O Delphi vem preparando o terreno para essa mudança desde a versão 3.

Os caracteres que conhecemos e estamos acostumados são os caracteres ASCII, caracteres formados por apenas 1 byte que podem assumir 256 valores diferentes. Na tabela ASCII encontramos, do 0 ao 127 (caracteres formados por apenas 7 bits) os caracteres de controle (enter, tab, backspace entre outros), as letras maiúsculas, minúsculas, números e símbolos. Dos valores 128 a 255 podemos encontrar a parte "estendida" do ASCII, que abrange cedilha e vogais acentuadas, mas que pode variar de acordo com a linguagem e o país. Esses são os caracteres de um byte que usávamos no Turbo Pascal e que continuamos usando até antes do Delphi 2009. Nossas Strings, Chars e PChars eram baseados nesses caracteres.

Desde o Delphi 3 existiam as WideStrings, formadas por WideChars. Os WideChars eram caracteres formados por dois bytes cada um, que poderiam assumir 65536 valores, podendo conter uma gama maior de caracteres, incluindo caracteres de língua estrangeira.

O Delphi 2005 já permitia que o código fonte fosse salvo em utf-8, e o Delphi 2007 já possuía drivers para dbExpress que funcionavam com Unicode. Além disso, desde o Delphi 6 era possível usar Unicode graças aos componentes TNT, um conjunto de controles como os da VCL que, usando WideStrings, permitia a entrada de caracteres de várias linguagens.

O que é Unicode

Unicode é um consórcio ou acordo mundial cujo objetivo é criar padrões para suportar múltiplos caracteres e linguagens nos computadores e softwares.

As 128 posições "altas" do ASCII simplesmente não funcionariam, porque há linguagens que necessitam de mais caracteres do que os 128 restantes, além de símbolos de pontuação e outros símbolos usados em matemática e filosofia. Além disso, o número de padrões diferentes para a parte "alta" do ASCII criou sim uma despadronização.

Existem vários "sabores" de implementação de Unicode, são os chamados Unicode Transformation Format (Formato de Transformação Unicode) ou UTF. Os padrões de UTF são: UTF-7, UTF-8, UTF-16, UTF-32. Os padrões UTF-16 e UTF-32 podem ainda ser Big Endian ou Litle Endian, referindo-se à orientação que um caractere composto por vários bytes deve ser lido: a partir da direita ou a partir da esquerda.

O UTF-7 faz parte de um conjunto de formatos obsoletos ou que tiveram de ser criados para necessidades específicas, mas não fazem parte do padrão.

O UTF-32 é o que desperdiça maior espaço, visto que todos os caracteres são formados por 4 bytes, de largura fixa, mesmo os "baixos" #0 até #127 compatíveis com ASCII. Apesar disso, o UTF-32 é o que apresenta a maior facilidade de conter todos os tipos de caracteres, de qualquer linguagem do mundo com nenhum processamento para isso. O UTF-32 é muito usado no mundo UNIX.

Já no padrão UTF-16 todos os caracteres têm 16 bits, ou 2 bytes. Isso nos daria a possibilidade de trabalharmos com até 65536 caracteres, o que representa apenas uma fração dos caracteres existentes no mundo inteiro. Felizmente o UTF-16 é expansível: ele permite caracteres de tamanho variável (podendo ter ou dois ou quatro bytes) com o uso de caracteres compostos e Surrogate Pairs, como veremos mais a frente. Uma desvantagem é que caracteres do #0 ao #255, de apenas um byte, serão representados por dois bytes: um idêntico ao que era na tabela ASCII e o outro será um #0. O UTF-16 é o mais usado nas APIs dos sistemas operacionais, máquinas virtuais JAVA, .Net Framework e por isso foi adotado para o Delphi.

O UTF-8 é o mais versátil, compacto e o mais usado na WEB. No UTF-8 um caractere pode ter 1, 2 ou até 4 bytes, dependendo de alguns valores "de corte" dos bytes menos significativos (mais a esquerda). Por exemplo, um caractere de #0 até #127 é representado por 1 byte, do #128 até o #2047 passa a ser representado por 2 bytes, do #2048 até o #65535 passa a ser representado por 3 bytes e do #65536 em diante passa a ser representado por 4 bytes. O UTF-8 não apresenta os problemas de desperdício de espaço com caracteres nulos que o UTF-16 e o UTF-32 apresentam.

No Delphi 2010 trabalharemos com os caracteres no formato UTF-16. Isso porque tanto o Delphi como a API do Windows podem trabalhar com WideStrings. O mapeamento de caracteres de 16 bits para WideStrings é, além de mais fácil, natural e transparente. O UTF-16 desperdiça menos espaço que o UTF-32, também pode ser expansível e o Delphi possui o tipo de String UTF8String e o Encoding UTF-8 caso seja necessário salvar arquivos nesse formato.

Por ser mais usado, o UTF-16 geralmente é chamado simplesmente de Unicode, inclusive no próprio Delphi e no .Net Framework, mas como vimos, Unicode vai muito além disso.

O que mudou

Com as mudanças uma coisa que não existe mais é o conceito de caractere. Pelo menos não como o conhecemos. Isso porque em um ambiente Unicode os conceitos que trabalhamos são o de Code Unit e Code Point, ou "unidade de código" e "ponto de código" em uma tradução livre e literal.

Além disso, temos os chamados Surrogate Pairs. Um Surrogate é um tipo de caractere, utilizado em algumas linguagens, que altera ou substitui um grupo de dois caracteres. Um Surrogate Pair com certeza tem mais de 2 bytes. Além disso, devido à diversidade de linguagens e regras linguísticas agora suportadas, também existe

o conceito de caractere composto. Este pode ter o tamanho de dois caracteres, ou seja, 4 bytes, mas ser exibido como um caractere só. São dois caracteres visualizados como um.

Neste artigo precisaremos de alguma versão do Delphi anterior à 2009, por exemplo, o popular Delphi 7, e precisaremos também de uma versão mais nova, como o Delphi 2010. Criaremos, em ambas as versões, um programa Win32 simples, que contém um Form, um Memo ocupando quase todo o tamanho do Form e um espaço para colocarmos botões. Nos botões faremos testes e imprimiremos mensagens no Memo. Depois de feitos alguns testes, vamos criar no Delphi 7 uma dll com métodos cujos argumentos são PChars, a serem lidos/escritos dentro da dll, e tentaremos usar essa dll no Delphi 2010. Nosso programa de teste, a ser feito tanto no Delphi 7 como no 2010 será como mostra a **Figura 1**.

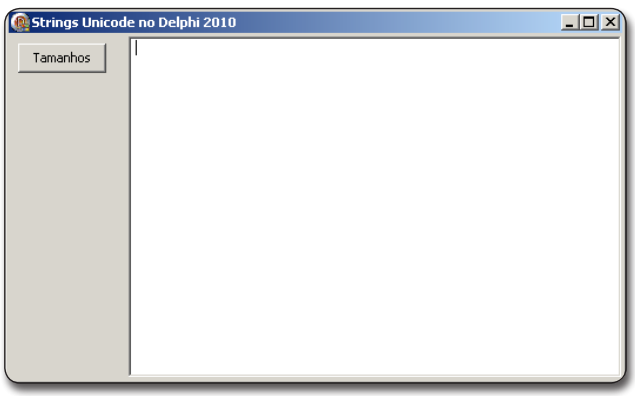


Figura 1. Formulário da aplicação de exemplo

O código da **Listagem 1** foi feito para ser testado no Delphi 2010. Nele temos uma variável String e uma AnsiString. Atribuiremos a ambas uma String formada pelos caracteres 'c', 'ç' e um caractere tirado aleatoriamente de um site japonês, por exemplo: 動. O que queremos demonstrar é que, além da diferença de tamanho dos caracteres, nem sempre as coisas no Delphi 2010 irão funcionar como funcionavam nas versões anteriores.

Ao clicarmos no botão, o resultado mostrado no Memo será o seguinte:

```
String: cç動
Tamanho do caracter c em uma string: 2 Número do caracter: 99
Tamanho do caracter ç em uma string: 2 Número do caracter: 231
Tamanho do caracter 動 em uma string: 2 Número do caracter: 21205
String: cç?
Tamanho do caracter c em uma string: 1 Número do caracter: 99
Tamanho do caracter ç em uma string: 1 Número do caracter: 231
Tamanho do caracter ? em uma string: 1 Número do caracter: 63
```

Como podemos notar, a String formada por caracteres de um byte ainda existe, e pode ser usada normalmente. Trata-se da AnsiString. A AnsiString não suportará o caractere 動 e não o exibirá corretamente, exibindo um ponto de interrogação no lugar. Além disso, seu número, obtido com a função ORD será o número 63, correspondente ao caractere "?". Cada caractere da String "normal" tem dois bytes, enquanto que da AnsiString só tem um.

Listagem 1. Mostrando tamanho e número dos caracteres em uma String

```
procedure TfrmTesteStrings.btTamanhosClick(Sender: TObject);
var
  UmaString: String;
  UmaAnsiString: AnsiString;
begin
  UmaString := 'cç動';
  UmaAnsiString := 'cç動';
  mTeste.Lines.Append('String: ' + UmaString);
  mTeste.Lines.Append('Tamanho do caracter '+
    UmaString[1] +' em uma string: ' + intToStr(sizeof( UmaString[1])) +
    ' Número do caracter: ' + IntToStr(ord(UmaString[1])));

  mTeste.Lines.Append('Tamanho do caracter '+
    UmaString[2] +' em uma string: ' + intToStr(sizeof( UmaString[2])) +
    ' Número do caracter: ' + IntToStr(ord(UmaString[2])));

  mTeste.Lines.Append('Tamanho do caracter '+
    UmaString[3] +' em uma string: ' + intToStr(sizeof( UmaString[3])) +
    ' Número do caracter: ' + IntToStr(ord(UmaString[3])));

  mTeste.Lines.Append('String: ' + UmaAnsiString);
  mTeste.Lines.Append('Tamanho do caracter '+
    UmaAnsiString[1] +' em uma string: ' + intToStr(sizeof( UmaAnsiString[1])) +
    ' Número do caracter: ' + IntToStr(ord(UmaAnsiString[1])));

  mTeste.Lines.Append('Tamanho do caracter '+
    UmaAnsiString[2] +' em uma string: ' + intToStr(sizeof( UmaAnsiString[2])) +
    ' Número do caracter: ' + IntToStr(ord(UmaAnsiString[2])));

  mTeste.Lines.Append('Tamanho do caracter '+
    UmaAnsiString[3] +' em uma string: ' + intToStr(sizeof( UmaAnsiString[3])) +
    ' Número do caracter: ' + IntToStr(ord(UmaAnsiString[3])));

end;
```

Nota

Para ter acesso a todos os métodos que funcionam com AnsiStrings devemos colocar a unit AnsiStrings no nosso uses.

No Delphi 7 esse mesmo código deveria falhar de qualquer forma. Podemos tentar adicionar uma variável do tipo Wide String ao Delphi 7, e o código da **Listagem 2**, mas mesmo assim o caractere não será exibido corretamente no Memo, por dois motivos:

1 – O Memo comum da VCL no Delphi 7 não tem suporte a Unicode/WideStrings;

2 – O Source Editor nesta versão não consegue colocar caracteres Unicode em seu arquivo .pas.

Faremos na **Listagem 3** alguns testes com UpperCase, usando a mesma String do exemplo anterior para vermos os resultados. E faremos o mesmo teste no Delphi 7.

O que esperaríamos como resultado dessa função? Teoricamente que o c em maiúsculo fosse C e o ç fosse Ç. Já o caractere 動 não teria sentido pois, trata-se de uma linguagem em que não existem caracteres maiúsculos e minúsculos. Entretanto, vejamos a saída do Memo:

```
String: cç動 em MAIÚSCULAS Cç動
caracter c em maiúsculo: C
caracter ç em maiúsculo: ç
caracter 動 em maiúsculo: 動
String: cç? em MAIÚSCULAS Cç?
```

```
caracter c em maiúsculo: C
caracter ç em maiúsculo: ç
caracter ? em maiúsculo: ?
String: cç動 em MAIÚSCULAS Cç動
caracter c em maiúsculo: C
caracter ç em maiúsculo: ç
caracter 動 em maiúsculo: 動
```

Listagem 2. Um teste com WideStrings no Delphi 7

```
var
  UmaWideString: WideString;

{...}

  UmaWideString := 'çç?';
  mTeste.Lines.Append('String: ' + UmaWideString);
  mTeste.Lines.Append('Tamanho do caracter '+
    UmaWideString[1] +' em uma string: ' + intToStr(sizeof( UmaWideString[1])) +
    ' Número do caracter: ' + IntToStr(ord(UmaWideString[1])));

  mTeste.Lines.Append('Tamanho do caracter '+
    UmaWideString[2] +' em uma string: ' + intToStr(sizeof( UmaWideString[2])) +
    ' Número do caracter: ' + IntToStr(ord(UmaWideString[2])));

  mTeste.Lines.Append('Tamanho do caracter '+
    UmaWideString[3] +' em uma string: ' + intToStr(sizeof( UmaWideString[3])) +
    ' Número do caracter: ' + IntToStr(ord(UmaWideString[3])));
```

Listagem 3. Teste com UpperCase

```
procedure TfrmTesteStrings.btUpperClick(Sender: TObject);
var
  UmaString: String;
  UmaAnsiString: AnsiString;
  UmaWideString: WideString;
begin
  UmaString := 'cç動';
  UmaAnsiString := 'cç動';
  UmaWideString := 'cç動';

  mTeste.Lines.Append('String: ' + UmaString + ' em MAIÚSCULAS ' +
  UpperCase(UmaString));
  mTeste.Lines.Append('caracter '+ UmaString[1] +' em maiúsculo: ' +
  UpperCase( UmaString[1]));
  mTeste.Lines.Append('caracter '+ UmaString[2] +' em maiúsculo: ' +
  UpperCase( UmaString[2]));
  mTeste.Lines.Append('caracter '+ UmaString[3] +' em maiúsculo: ' +
  UpperCase( UmaString[3]));

  mTeste.Lines.Append('String: ' + UmaAnsiString + ' em MAIÚSCULAS ' +
  UpperCase(UmaAnsiString));
  mTeste.Lines.Append('caracter '+ UmaAnsiString[1] +' em maiúsculo: ' +
  UpperCase( UmaAnsiString[1]));
  mTeste.Lines.Append('caracter '+ UmaAnsiString[2] +' em maiúsculo: ' +
  UpperCase( UmaAnsiString[2]));
  mTeste.Lines.Append('caracter '+ UmaAnsiString[3] +' em maiúsculo: ' +
  UpperCase( UmaAnsiString[3]));

  mTeste.Lines.Append('String: ' + UmaWideString + ' em MAIÚSCULAS ' +
  UpperCase(UmaWideString));
  mTeste.Lines.Append('caracter '+ UmaWideString[1] +' em maiúsculo: ' +
  UpperCase( UmaWideString[1]));
  mTeste.Lines.Append('caracter '+ UmaWideString[2] +' em maiúsculo: ' +
  UpperCase( UmaWideString[2]));
  mTeste.Lines.Append('caracter '+ UmaWideString[3] +' em maiúsculo: ' +
  UpperCase( UmaWideString[3]));

end;
```

Por incrível que pareça, a função UpperCase usada em ç retorna ç, em vez de Ç. Isso pode ser decepcionante, afinal, deve haver centenas de lugares em nosso código onde usamos UpperCase para gravar tudo em maiúsculas no banco de dados, ou LowerCase para gravar tudo em minúscula, ou, frequentemente, convertemos Strings para maiúsculas ou minúsculas para compararmos. Infelizmente teremos que vasculhar em nosso código fonte para alterarmos essas chamadas.

Nota do DevMan

Nova UnicodeString x WideString

Uma String nos Delphi's anteriores ao 2009 era um ponteiro para uma estrutura que possuía 4 bytes para contagem de referência, 4 bytes para marcar o tamanho, e um vetor de até 216 caracteres de um byte cada um. Enquanto isso, as WideStrings eram Strings sem contagem de referência, por isso mais lentas, mas eram formadas por caracteres de 2 bytes cada um. Antigamente as AnsiStrings eram a mesma coisa que os Strings. Na nova versão do Delphi as AnsiStrings continuam existindo, com contagem de referência e tudo o mais, as WideStrings também continuam existindo, sem a contagem de referências, e a String é um novo tipo de estrutura, chamada de UnicodeString, existente desde o Delphi 2007 mas agora "mapeada" para o nome String. Esse novo tipo não é igual ao WideString, mas é "Reference Counted" e possui quatro bytes a mais, em sua estrutura, 2 para armazenar o "codepage" e 2 para armazenar o "element size" (tamanho de um caractere).

Um programador Delphi mais experiente pode argumentar que esse resultado era perfeitamente esperado, já que no Delphi 7 a função UpperCase também não conseguia transformar o ç em maiúscula. Para fazer isso era necessário usar a função AnsiUpperCase, como mostra a **Listagem 4**.

Um ponto de confusão no Delphi 7 e 2010 é que poderíamos usar a função AnsiUpperCase em uma WideString, ou usar WideUppercase em uma AnsiString. Mesmo assim, como esperado, a função AnsiUpperCase retorna "Ç" para o caractere "ç".

Entretanto, no Delphi 2010, para retornar Ç devemos também usar AnsiUpperCase. Isso pode soar como uma inconsistência, afinal, estamos usando um método Ansi para converter Strings Unicode. Esperamos que nas próximas versões do Delphi, a Embarcadero crie uma documentação melhor sobre o assunto, algumas Warnings avisando sobre o uso das funções Ansi com caracteres Unicode e talvez até tornar algumas funções "Deprecated" e criar novas, para que esteticamente nosso código fique mais "elegante". Enquanto isso você pode usar as funções Wide para as Strings nativas do Delphi 2010, do tipo UnicodeStrings, já que elas são mais "aparentadas" com as WideStrings.

O mesmo código da **Listagem 4** funcionará perfeitamente no Delphi 2010. Isso porque a função UpperCase é feita nativamente no Delphi, em pascal puro, enquanto que as funções AnsiUpperCase e WideUperCase são mapeadas da API do Windows, pelas funções CharUpperBuffA e CharUpperBuffW respectivamente.

Um outro exercício interessante que podemos fazer é uma tabela gigante Unicode. Geralmente, quando precisávamos de uma tabela ASCII, desde o Turbo Pascal, usávamos um código semelhante ao da **Listagem 5** para imprimir a tabela.

Listagem 4. Teste com AnsiUpperCase no Delphi 7

```
procedure TForm1.Button1Click(Sender: TObject);
var
  UmaString: String;
  UmaAnsiString: AnsiString;
  UmaWideString: WideString;
begin
  UmaString := 'çç?';
  UmaAnsiString := 'çç?';
  UmaWideString := 'çç?';

  mTeste.Lines.Append('String: ' + UmaString + ' em MAIÚSCULAS ' +
  AnsiUpperCase(UmaString));
  mTeste.Lines.Append('caracter '+ UmaString[1] +' em maiúsculo: ' +
  AnsiUpperCase( UmaString[1]));
  mTeste.Lines.Append('caracter '+ UmaString[2] +' em maiúsculo: ' +
  AnsiUpperCase( UmaString[2]));
  mTeste.Lines.Append('caracter '+ UmaString[3] +' em maiúsculo: ' +
  AnsiUpperCase( UmaString[3]));

  mTeste.Lines.Append('String: ' + UmaAnsiString + ' em MAIÚSCULAS ' +
  AnsiUpperCase(UmaAnsiString));
  mTeste.Lines.Append('caracter '+ UmaAnsiString[1] +' em maiúsculo: ' +
  AnsiUpperCase( UmaAnsiString[1]));
  mTeste.Lines.Append('caracter '+ UmaAnsiString[2] +' em maiúsculo: ' +
  AnsiUpperCase( UmaAnsiString[2]));
  mTeste.Lines.Append('caracter '+ UmaAnsiString[3] +' em maiúsculo: ' +
  AnsiUpperCase( UmaAnsiString[3]));

  mTeste.Lines.Append('String: ' + UmaWideString + ' em MAIÚSCULAS ' +
  AnsiUpperCase(UmaWideString));
  mTeste.Lines.Append('caracter '+ UmaWideString[1] +' em maiúsculo: ' +
  AnsiUpperCase( UmaWideString[1]));
  mTeste.Lines.Append('caracter '+ UmaWideString[2] +' em maiúsculo: ' +
  AnsiUpperCase( UmaWideString[2]));
  mTeste.Lines.Append('caracter '+ UmaWideString[3] +' em maiúsculo: ' +
  AnsiUpperCase( UmaWideString[3]));
end;
```

Nota do DevMan

Se ao realizar a migração de algum projeto seu desenvolvido nas versões anteriores ao 2010 e se deparar com alguma mensagem "Deprecated", você deve ficar atento àquele código. Esta mensagem refere-se a um determinado método que está em uso no seu sistema, mas que sofreu uma "atualização", sendo necessário compatibilizá-lo com o novo método para que esta mensagem suma. É comum notar esta mensagem caso você trabalhe com o dbExpress, isto pelas diferenças da versão do DBX no Delphi 7, Delphi 2007 e Delphi 2009, na qual o Framework sofreu atualizações.

Nota do DevMan

Uma mudança importante na unit Windows é que, antigamente, as funções sem terminação A ou W eram todas mapeadas para a função Ansi da API, terminada em A, mas agora, as funções sem terminação são mapeadas para os métodos correspondentes terminados em W. Mesmo assim a unit Windows sempre manterá as duas versões das funções, terminadas em A ou W.

Nota

O método WideUpperCase também deverá ser capaz de transformar "ç" em "Ç" em ambas as versões do Delphi.

Listagem 5. Tabela ASCII

```
procedure TForm1.btTabelaClick(Sender: TObject);
var i: integer;
begin
  for i := 0 to 255 do
    mTeste.Lines.Append(inttostr(i) +  #9 + chr(i) );
end;
```

Agora podemos fazer uma tabela UTF-16 de 65536 caracteres com o código da **Listagem 6**. Note que os 256 primeiros caracteres, do 0 a 255, são iguais em ambas as tabelas, se a fonte for a mesma. Porém, do caractere 256 em diante podemos ver caracteres japoneses, chineses, coreanos, gregos e assim por diante.

Listagem 6. Tabela UTF-16

```
procedure TForm1.btTabelaClick(Sender: TObject);
var i: integer;
begin
  for i := 0 to 65535 do
    mTeste.Lines.Append(inttostr(i) +  #9 + chr(i) );
end;
```

Obviamente essa tabela UTF-16 é apenas um arranhão em todo o universo Unicode, porque contém apenas os 65536 caracteres que podem ser mapeados "diretamente", mas não contém os caracteres compostos ou Surrogate Pairs.

Se o código da **Listagem 6**, com 65536 caracteres, for usado no Delphi 7, simplesmente a tabela se repetirá inteira de 256 em 256 caracteres, aparecendo a mesma tabela 256 vezes.

Migrando para Unicode

Teoricamente a migração será bem fácil, sem maiores dores de cabeça. Mas temos alguns casos que deverão ser observados com cuidado:

1 - Warnings de conversão implícita.

Cada uma dessas Warnings deve ser tratada caso a caso. Devemos analisar se não estamos gravando caracteres de uma outra linguagem em uma variável AnsiString e se isso não causará perda de dados. Ao gravar uma AnsiString em uma variável do tipo String podemos usar a conversão explícita para evitar Warnings.

2 - I/O.

Arquivos gravados em disco, comunicação TCP/IP, streams e buffers devem ser revistos. Os tamanhos dos buffers devem ser aumentados e a maneira como definimos tamanhos de chars e strings deve ser revista.

3 – Tamanhos.

Nunca assuma que um Char tem um byte, e nem assuma que tem 2, pois como vimos, o tamanho de um Char agora pode variar. Então buffers de bytes não são mais válidos para se transformar Strings em PChars para se passar à API's externas. Mais importante ainda, nunca assuma que o comprimento de uma String seja igual ao número de bytes dela, ou que seu tamanho em bytes tenha relação com o número de caracteres.

4 - API's externas, dll's de versões anteriores do Delphi.

Se não for possível alterar as dll's para que fiquem com funções terminadas em W para WideStrings e A para AnsiStrings, então pelo menos teremos de tentar converter nossas Strings para AnsiString. Então, em todos os pontos em que nosso sistema "toca" um ambiente ou aplicação externo, temos de nos certificar de que estamos passando dados do tipo e tamanho corretos.

5 - Sets de caracteres.

Os sets continuam suportando somente AnsiChars. Então, dentro de um set, um caractere passa a ser automaticamente convertido para AnsiChar. Para verificar se um caractere está em um conjunto, use o método CharInSet(c, ['a', 'b', 'c']) no lugar de if c in ['a', 'b', 'c'].

6 - Uso de CHR para obter um Char a partir de um inteiro.

No Delphi 2010 a função CHR retorna um WideChar de dois bytes, e pode aceitar como parâmetro valores maiores que 255, assim como as conversões Char(x) e WideChar(x). Para obter um AnsiChar correspondente a algum número de #0 a #255 (8 bits) utilize a conversão AnsiChar(x).

Se em sistemas legados você antes usava CHR para obter um caractere através de um número, tome o cuidado agora de substituir o CHR(x) por AnsiChar(x), porque o CHR irá retornar um caractere de 2 bytes.

7 - Uso de FillChar.

Sempre que usamos vetores de Chars para serem passados como PChars para APIs usamos o método FillChar para preencher todo o espaço disponível com o caractere #0 ou null. A maneira como usamos FillChar deve ser alterada. A **Listagem 7** mostra como fazíamos antes e como devemos fazer agora.

Listagem 7. Uso de FillChar

```
var
  qtde: Integer;
  Buffer: array[0..255] of Char;
begin
  // Como fazíamos antes
  qtde := Length(Buffer);
  FillChar(Buffer, qtde, 0);

  // como devemos fazer agora
  qtde := Length(Buffer) * SizeOf(Char); //
  <<-- especificando o tamanho de um vetor de chars de maneira correta.
  FillChar(Buffer, qtde, 0);
end;
```

8 - Chamadas ao método Move.

Da mesma forma que o FillChar, devemos agora levar em consideração o tamanho real de uma String em bytes, e não seu comprimento. A **Listagem 8** ilustra a mudança.

9 - Comparações entre caracteres literais.

Veremos mais adiante que comparações entre Strings, ou de uma string vinda de um Edit, ou de um banco de dados com uma constante podem não ter o resultado que esperamos, devido a problemas com caracteres similares ou caracteres acentuados, que podem ser pré-compostos ou compostos. Mostraremos como podemos contornar essa situação.

Listagem 8. Uso de Move

```
var
  qtde: Integer;
  Buf1, Buf2: array[0..255] of Char;
begin
  // Como fazíamos antes
  qtde := Length(Buf1);
  Move(Buf1, Buf2, qtde);

  // Como devemos fazer agora, com UnicodeString
  qtde := Length(Buf1) * SizeOf(Char); // <<-- tamanho correto, em bytes.
  Move(Buf1, Buf2, qtde);
end;
```

10 - Alocação de memória para ponteiros PChar.

Você deve ter em mente que agora o PChar aponta para um tipo de dado que tem o dobro do tamanho de antes, então toda sua aritmética de ponteiros e sua alocação dinâmica deve ser revista. Ao trabalhar com bibliotecas legadas, use PAnsiChar e AnsiChar no lugar dos PChar e Char. Você pode usar msg := StrNew(PAnsiChar('Clube Delphi 2010')); para alocar dinamicamente um PAnsiChar com o valor 'ClubeDelphi'. Sem esquecer de usar o StrDispose depois, para liberar a memória.

Sempre que essas dez situações estiverem envolvidas com bancos de dados, TFields tipados, com componentes de terceiros, arquivos, dll's de terceiros e operações com ponteiros, a atenção deve ser redobrada. Ao longo desse artigo veremos como solucionar alguns desses dez desafios.

Uma ferramenta chamada Unicode Statistics Tool pode ser baixada no Code Central da Embarcadero para nos auxiliar a avaliar a complexidade de um programa quanto à sua migração para Unicode. A **Figura 2** mostra a ferramenta em uso. Essa ferramenta faz uma análise no seu código fonte e mostra em que units estão localizados os pontos problemáticos a serem tratados.

Migrando dll's

Para esse exemplo vamos criar uma dll em Delphi 7 com algumas funções. Basicamente criaremos uma função que mostra uma mensagem e outra que modifica uma mensagem. Consumiremos os métodos da dll pelo Delphi 7, e depois tentaremos consumi-los pelo Delphi 2010. Com isso iremos mostrar os problemas que podemos enfrentar e como contorná-los ao migrar nossa aplicação antiga ao universo Unicode, mesmo usando dll's de terceiros legadas. Essas dll's são do tipo que trabalham com PChar, com convenção de chamada stdcall, para serem iguais as dll's da API do Windows. Esse tipo de DLL pode ser encontrada em produtos feitos em Delphi ou qualquer versão de C/C++ para Windows. Vamos criar no Delphi 7 a dll mostrada na **Listagem 9**.

Esta dll que fizemos possui duas funções: EchoMessage que simplesmente mostra uma mensagem usando o método MessageBox da API do Windows, e a RetornaData que retornará uma saudação contendo a data.

Vamos criar no Delphi 7 o código para consumir esses métodos e vamos observar seu comportamento. Observe o código na **Listagem 10.**

Listagem 9. Código para a criação da dll

```
library TesteDll;

{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL--even those that
  are nested in records and classes. ShareMem is the interface unit to
  the BORLNDMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using BORLNDMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  Windows,
  SysUtils;

{$R *.res}

procedure EchoMessage(msg: PChar); stdcall;
begin
  MessageBox(0, msg, 'Mensagem:', 0);
end;

procedure RetornaData(msg: PChar); stdcall;
var
  strData: string;
begin
  strData := DateTimeToStr(now);
  StrCopy(msg, PChar('Olá! Hoje é :' + strData));
end;

exports
  EchoMessage,
  RetornaData;

begin
end.
```

Listagem 10. Consumindo a dll

```
procedure EchoMessage(msg: PChar); stdcall external 'TesteDll.dll' name
'EchoMessage';

procedure RetornaData(msg: PChar); stdcall external 'TesteDll.dll' name
'RetornaData';

{...}

procedure TForm1.btTesteDllClick(Sender: TObject);
var
  msg : array[1..255] of Char;
begin
  EchoMessage('Clube Delphi 2010');
  RetornaData(@msg);
  MessageBox(0, @msg, 'Data', 0);
end;
```

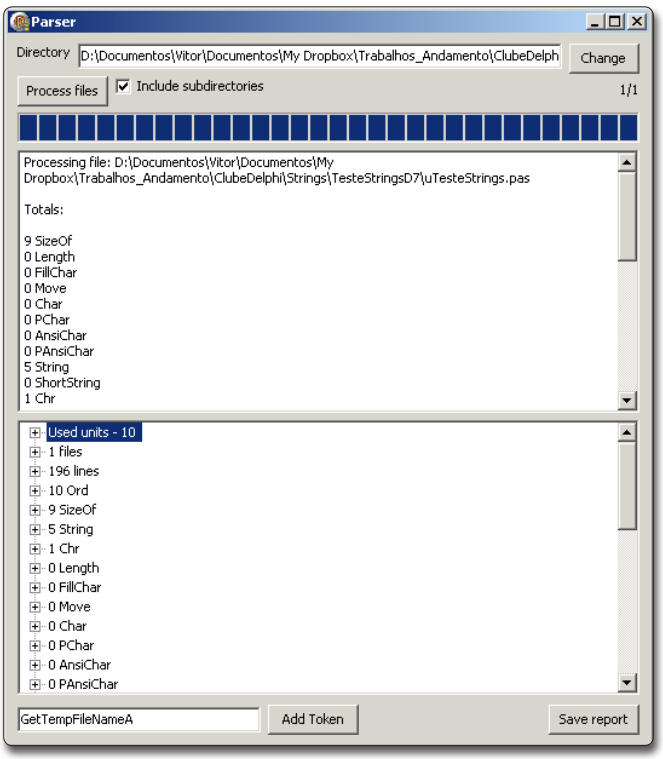


Figura 2. Usando a ferramenta Unicode Statistics Tool

Figura 3. Mensagem gerada pelo método EchoMessage

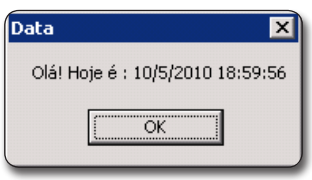


Figura 4. Mostrando o resultado de RetornaData

Figura 5. Mensagem gerada pelo método EchoMessage

Figura 6. Mostrando o resultado de RetornaData

Executando o código receberemos, corretamente, as mensagens vistas nas **Figuras 3** e **4**. Não há nenhum segredo até agora.

Agora vamos colocar esse mesmo código no Delphi 2010 e tentar executar os métodos, para ver o que obteremos, nas **Figuras 5** e **6**.

Nenhum dos dois funcionou como deveria, mas por motivos diferentes. O EchoMessage, na dll é padrão ANSI. Ele espera que seja enviada uma String terminada em #0 para mostrar com o MessageBox. O MessageBox mostrará os caracteres até encontrar o #0.

Como os caracteres Unicode, no Delphi 2010, têm 2 bytes cada um, o "C" de "Clube Delphi" em vez de ser apenas #67 é formado por #67#0, e nossa String ANSI considera isso como sendo final de String. Por isso apenas o "C" é mostrado. As WideStrings e UnicodeStrings terminam em #0#0 em vez de só #0.

Já com o método RetornaData ocorre o contrário. As Strings são montadas dentro da dll com vários caracteres de um único byte, mas ao passar esse ponteiro para o Delphi 2010 e ao tentar exibir a String, os bytes são agrupados de dois em dois, gerando caracteres diferentes, até que seja encontrado, em algum lugar da memória, um #0#0. Se ele não for encontrado então a mensagem poderá até trazer lixo aleatório da memória, ultrapassando o ponto onde teoricamente deveria acabar.

Para usar a dll ANSI corretamente a partir do Delphi 2010 temos de fazer algumas alterações. A **Listagem 11** ilustra o novo código.

Devemos levar em conta que no Delphi 7 a API MessageBox é mapeada para MessageBoxA, enquanto que no Delphi 2010 a mesma é mapeada para MessageBoxW. Note que nesses casos sempre teremos que fazer uma dupla conversão. Na linha onde temos o código: MessageBox(0, PWideChar(string(data)), 'Data', 0); data já é um vetor de AnsiChars, implicitamente conversível para PAnsiChar e AnsiString, então a conversão para PAnsiChar e AnsiString aqui pode ser omitida. Porém, a função MessageBox[W] necessita de um PWideChar, por isso a String é convertida para o tipo String (UnicodeString é facilmente mapeada para WideString) e depois convertida para um ponteiro PWideChar. Sempre necessitaremos dessa dupla-conversão para transformar AnsiStrings em UnicodeStrings e por sua vez em PChar.

Para que possamos fazer com que uma dll desenvolvida para Delphi 7 funcione no Delphi 2010 devemos obrigatoriamente mapear as funções da dll que usam PChar para PAnsi Char, e as que usam Char para AnsiChar. Os parâmetros a serem passados e os retornos devem ser todos baseados em AnsiString e AnsiChar.

Lendo e gravando arquivos texto

Vamos fazer aqui dois testes: usar o método SaveToFile do Memo no Delphi 7 e abrir o arquivo salvo com o programa desenvolvido em Delphi 2010, usando LoadFromFile, e depois tentar fazer o inverso. O que veremos sobre os métodos SaveToFile e LoadFromFile do Memo deve se aplicar aos métodos de mesmo nome de qualquer componente da VCL. Vamos adicionar mais dois botões na nossa aplicação, intitulados "Abrir" e "Salvar". O código da **Listagem 12** mostra como salvar ou carregar arquivos para um TMemo usando TOpenDialog ou TSaveDialog criados em Runtime.

Se você inserir alguns caracteres Unicode no Memo e salvar, verá que o arquivo foi salvo em ASCII. De fato ASCII é o Encoding padrão para ler ou gravar arquivos, porém, os caracteres estrangeiros não serão suportados e serão trocados por um "?".No Delphi 2010 existe um segundo método SaveToFile sobrecarregado com um segundo parâmetro chamado TEncoding. Podemos usar as propriedades estáticas da classe TEncoding como mostra a **Listagem 13**.

Na hora de abrir o arquivo o método LoadFromFile, o Memo detectará automaticamente o tipo de arquivo e configurará o componente. Mesmo assim podemos, e devemos, explicitar o Encoding que vamos usar. Faça os testes de leitura e escrita também com o Encoding UTF-8.

Novas noções de comprimento

Existem dois conceitos básicos em Unicode que devem ser entendidos: o conceito de Code Unit e o de Code Point. Code Unit é cada unidade de dois bytes necessária para formar um caractere. Code Point é o caractere em si. Surrogate Pair é quando dois Code Units formam um único Code Point, ou seja, uma letra "formada" por dois caracteres. Quando uma string contém um Surrogate Pair o método Length retornará um resultado que pode parecer incorreto. Isso porque ele contará o Surrogate Pair como dois caracteres.

Listagem 11. Consumindo a dll

```
procedure EchoMessage(msg: PAnsiChar); stdcall external 'TesteDll.dll' name
'EchoMessage';

procedure RetornaData(msg: PAnsiChar); stdcall external 'TesteDll.dll' name
'RetornaData';

{...}

procedure TfrmTesteStrings.btTesteDLLClick(Sender: TObject);
var
  data : array[1..255] of AnsiChar;
  msg: AnsiString;
begin
  msg := 'Clube Delphi 2010';
  EchoMessage(PAnsiChar(msg));
  RetornaData(@data);
  MessageBox(0, PWideChar(string(data)), 'Data', 0);
  //ou
  MessageBoxA(0, @data, 'Data', 0);
end;
```

Listagem 12. Salvando e abrindo arquivos de Texto

```
procedure TfrmTesteStrings.btSalvarClick(Sender: TObject);
begin
  with TSaveDialog.Create(nil) do
  try
    DefaultExt := 'txt';
    Filter := 'Arquivos de texto (*.txt)|*.txt';
    if Execute then
      mTeste.Lines.SaveToFile(FileName);
  finally
    Free;
  end;
end;

procedure TfrmTesteStrings.btAbrirClick(Sender: TObject);
begin
  with TOpenDialog.Create(nil) do
  try
    DefaultExt := 'txt';
    Filter := 'Arquivos de texto (*.txt)|*.txt';
    if Execute then
      mTeste.Lines.LoadFromFile(FileName);
  finally
    Free;
  end;
end;
```

Listagem 13. Escolhendo um Encoding

```
procedure TfrmTesteStrings.btSalvarClick(Sender: TObject);
begin
  with TSaveDialog.Create(nil) do
  try
    DefaultExt := 'txt';
    Filter := 'Arquivos de texto (*.txt)|*.txt';
    if Execute then
      mTeste.Lines.SaveToFile(FileName, TEncoding.Unicode);
  finally
    Free;
  end;
end;
```

Nota do DevMan

BOM (byte order Mark) é um conjunto de 2 a 3 bytes reservados, gravados no início de um arquivo que diz para a aplicação qual é o "sabor" de Unicode em que ele foi gravado: UTF-7, UTF-8, UTF-16 BE, UTF-16 LE, UTF-32 BE e UTF-32 LE. Arquivos sem BOM geralmente são arquivos ASCII puros e podem ser carregados como Encoding ASCII. Nesses casos devemos omitir o segundo parâmetro do LoadFromFile, para que seja detectado ASCII. Assim o sistema se encarregará automaticamente de criar o "byte #0 extra" para cada byte gravado no arquivo. Caso contrário, se você tentar forçar o Encoding como Unicode, o sistema irá agrupar os bytes dos caracteres gravados de dois em dois transformando o texto em um monte de "Kanjis".

Se você estiver portando sua aplicação para alguma linguagem em que se usam esses tipos de caracteres, deve analisar bem os lugares onde usa a função Length para decidir qual a informação que você quer obter: o número de caracteres ou o número de letras de uma String. Sendo assim, nossa noção de comprimento agora vai além de saber quantas letras uma String aparenta ter, mas sim quantos caracteres ela realmente tem.

Na **Figura 7** podemos ver um caractere Kanji (ideograma) de numeração alta no projeto Unicode. Trata-se do caractere #$2A6D6 que pode ser obtido pelos caracteres #$D869 e #$DED6. Este é o último caractere da extensão B do Unicode para caracteres orientais (CJK Ideograph Extension B). Retiramos esse caractere da base unihan, do projeto Unicode, cujo endereço você pode ver na seção de links. Essa base fornece um serviço de busca e visualização de caracteres orientais. Muitos desses caracteres têm uma numeração ou índice altos, inclusive para os padrões Unicode UTF-16.

Na **Listagem 14** iremos compor uma String com sete "letras", onde as seis primeiras serão 'Teste', mais um espaço ou traço e seguido do kanji da **Figura 7**.

Figura 7. Kanji cujo valor unicode é 2A6D6, sendo seu Surrogate Pair D869 e DED6

É possível que esse caractere não seja visualizado corretamente em seu computador dependendo de seu suporte a idiomas e de suas fontes. Mas o importante é detectar a presença do mesmo, sendo visualizado corretamente ou não.

Para poder visualizar esse caractere você deverá instalar o suporte a idiomas do leste asiático do Windows e procurar na internet pela fonte Code2002. Há também a fonte LastResort que pode ser baixada no site do projeto Unicode. Esta fonte não mostra esse caractere corretamente, mas mostra em seu lugar um quadro com uma marcação dizendo que esse é o caractere 2A6D6.

Listagem 14. Length X ElementToCharLen com surrogate pairs

```
procedure TfrmTesteStrings.btTamSorrogateClick(Sender: TObject);
var
  UmaString: string;
begin
  UmaString := 'Teste-'#$d869#$ded6;
  mTeste.Lines.Add(UmaString);
  mTeste.Lines.Add('Retorno da função Length:'
    +IntToStr(Length(UmaString)));
  mTeste.Lines.Add('Retorno da função ElementToCharLen: '
    +IntToStr(ElementToCharLen(UmaString, 255)));
end;
```

Como saída da **Listagem 14** você terá:

```
Teste-𪚥
Retorno da função Length: 8
Retorno da função ElementToCharLen: 7
```

Isso significa que a função Length retornará 8, afinal essa String tem 8 Code Units, ou grupos de 2 bytes. Mas a função ElementToCharLen retornará apenas 7, porque essa String tem 7 Code Units. Este não é o único ponto de confusão a respeito do tamanho das Strings. Além dos Surrogates existem outros tipos de caracteres que são caracteres compostos. Eles aparecem como um único caractere, nos controles, para todos os efeitos. Se você tiver um caractere desses em um Edit, por exemplo, ao copiar e colar ele se comportará como um único caractere. Esse tipo de caractere pode ser formado por dois ou mais Code Units (caracteres), mas também conter dois ou mais Code Points, diferentemente dos Surrogate Pairs. Isso faria com que as funções Length e ElementToCharLen retornassem o mesmo tamanho, conforme o exemplo da **Listagem 15**.

Listagem 15. Length X ElementToCharLen com caracteres compostos

```
procedure TfrmTesteStrings.btTamCompostoClick(Sender: TObject);
var
  UmaString: string;
begin
  UmaString := 'teste-'+#$006f#$0308;//caractere ö
  mTeste.Lines.Add(UmaString);
  mTeste.Lines.Add('Retorno da função Length: '
    +IntToStr(Length(UmaString)));
  mTeste.Lines.Add('Retorno da função ElementToCharLen: '
    +IntToStr(ElementToCharLen(UmaString, 255)));

end;
```

Usaremos para esse exemplo a palavra 'teste' seguido de um traço e um caractere composto formando a string 'teste-o¨'. O caractere usado é uma espécie de letra "o" com trema (o¨), mas não é o "ö" tradicional, e sim um outro caractere de número maior. Isso deve ser explicado da seguinte forma: muitos dos caracteres no enorme panteão Unicode podem até ser graficamente parecidos, mas têm valores sintáticos e fonéticos totalmente diferentes, e seu número e posição na tabela também difere. Por exemplo, existe um caractere russo que parece muito o numeral "3", mas tem valor totalmente diferente.

Além disso, o padrão Unicode separa os glifos das letras e dos acentos diacríticos, para que eles sejam compostos através da combinação de dois caracteres. Porém os caracteres pré-

compostos, que já existiam antes do padrão Unicode, tiveram de ser mantidos por questões de compatibilidade. Por isso ficamos com alguns caracteres graficamente repetidos, e com valor semântico repetido também. Esse é o caso dos caracteres á, é, í, ó, ú, entre outros. Embora possamos pensar que essa String tem apenas 7 caracteres, contando o traço, na verdade tem 8, porque o último é composto. Ao clicarmos no botão receberemos o seguinte resultado no Memo:

```
teste-ö
Retorno da função Length: 8
Retorno da função ElementToCharLen: 8
```

As duas funções nos dizem que a String tem 8 caracteres, quando na verdade só podemos ver 7. Felizmente o Windows nos fornece uma API de normalização de Strings para lidar com esses casos. Ela funciona apenas com caracteres compostos e não com Surrogates. Na **Listagem 16** vamos criar as referências para as funções que precisaremos na dll 'Normaliz.dll' e vamos criar uma função que verifica se a String passada é normalizada. Se for, usa uma função da API para verificar o tamanho, caso contrário usa a função Length. Você pode usar ElementToCharLen no lugar de Length se quiser saber a quantidade de letras e suspeitar que sua String pode ter caracteres compostos e Surrogates misturados (pense numa tabela de fornecedores de vários países em um banco de dados de um sistema de importação e exportação). Principalmente se o que você deseja saber é o número de "letras" em uma String e não seu tamanho real em Code Units ou quanto espaço ela ocupa na memória.

Com esta função criada você pode adicionar ao código da **Listagem 15** essa última linha:

```
mTeste.Lines.Add('Retorno da função NormalizedStringLength: '+IntToStr(Normalize
dStringLength(UmaString)));
```

E clicar no botão, verificando o retorno.

```
teste-ö
Retorno da função Length: 8
Retorno da função ElementToCharLen: 8
Retorno da função NormalizedStringLength: 7
```

Como vimos, para essa String especificamente, a função NormalizedStringLength retorna 7 embora a Length e a ElementToCharLen retornem 8. Isso porque o caractere "ö" é um caractere composto e não um Surrogate.

Comparação de Strings

Para finalizar nosso artigo iremos comparar duas Strings que graficamente são iguais, mas que não são formadas pelo mesmo caractere. A **Listagem 17** mostrará como essa comparação foi feita.

No exemplo da **Listagem 17** voltamos a usar as duas variantes de "ö" que conhecemos. Usando o sinal de igual "=" para comparar as duas Strings somos notificados de que as Strings são diferentes, e realmente são. Mas usando WideCompareStr podemos ver que as duas Strings são iguais. WideCompareStr é uma função que compara duas Strings e retorna um número inteiro que pode ser:

```
< 0 se S1<S2.
0 se S1=S2.
> 0 se S1>S2.
```

Como resultado receberemos a mensagem:

```
hellö e hellö são diferentes
hellö e hellö são iguais pelo método WideCompareStr
```

Listagem 16. Função NormalizedStringLength

```
//funções para normalização de strings
const
  NormalizationOther  = 0;
  NormalizationC      = 1;
  NormalizationD      = 2;
  NormalizationKC     = 5;
  NormalizationKD     = 6;

function IsNormalizedString(NormForm: Integer; lpString: LPCWSTR;
  cwLength: Integer): BOOL; stdcall; external 'Normaliz.dll';

function NormalizeString(NormForm: Integer; lpSrcString: LPCWSTR;
  cwSrcLength: Integer; lpDstString: LPWSTR;
  cwDstLength: Integer): Integer; stdcall; external 'Normaliz.dll';

function NormalizedStringLength(const S: string): Integer;

implementation

{$R *.dfm}

function NormalizedStringLength(const S: string): Integer;
var
  Buf: string;
begin
  if not IsNormalizedString(NormalizationC, PChar(S), -1) then
  begin
    SetLength(Buf, NormalizeString(NormalizationC,
     PChar(S), Length(S), nil, 0));
    Result := NormalizeString(NormalizationC, PChar(S),
              Length(S), PChar(Buf), Length(Buf));
  end
  else
    Result := Length(S);
end;
```

Listagem 17. Comparando strings

```
procedure TfrmTesteStrings.btComparacaoClick(Sender: TObject);
var
  str1, str2: string;
begin

  str1 := 'hellö';
//literal a ser comparado. pode ser um comando ou opção de menu
  str2 := 'hell'#$006f#$0308;
//simulando um caractere estrangeiro vindo de um banco de dados ou input

  if str1 = str2 then
    mTeste.Lines.Append(str1 + ' e ' + str2 + ' são iguais')
  else
    mTeste.Lines.Append(str1 + ' e ' + str2 + ' são diferentes');

  if WideCompareStr(str1, str2)=0 then
    mTeste.Lines.Append(str1 + ' e ' + str2 + ' são iguais pelo método
WideCompareStr')
  else
    mTeste.Lines.Append(str1 + ' e ' + str2 + ' são diferentes pelo método
WideCompareStr');

end;
```

Nota

Existem as funções WideCompareStr e WideCompareText, a diferença entre as duas é que a WideCompareStr é Case Sensitive, enquanto a WideCompareText é Case Insensitive, levando em conta apenas o texto e não a caixa. Analogamente a estas temos as funções AnsiCompareStr e AnsiCompareText. Além disso, temos as funções AnsiSameText e WideSameText que comparam duas Strings independente da caixa e retornam Boolean em vez de Integer.

Também existe a função AnsiCompareStr que fará o mesmo com Strings ANSI. Por isso, qualquer tentativa de validar uma entrada, ou um menu, ou uma configuração em um arquivo comparando com o literal 'hellö' pode falhar dependendo de que "ö" for usado. Usamos esse caractere como exemplo, mas há outros na mesma situação.

Conclusão

Esperamos que com esse artigo possamos tornar a migração para Unicode fácil e natural. Além das técnicas apresentadas aqui temos uma lista na seção links muito útil, e dentre os links temos um pdf da Embarcadero, intitulado "Delphi Unicode Migration for Mere Mortals: Stories and Advice from the Front Lines" que é de leitura obrigatória.

Unicode não é a garantia de que seu software será portado para outros idiomas e países, mas é o primeiro passo. Como vimos, o maior passo do Delphi nessa direção foi adotar o tipo UnicodeString como a String nativa. É um grande passo da Embarcadero nessa direção, e esperado já há um bom tempo pelos desenvolvedores Delphi.

Vimos também que o uso de muitos métodos que estamos acostumados a usar com Strings, terão de ser adaptados, e alguns até perderam o sentido. Esperamos que isso seja melhorado nas próximas versões do Delphi. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Links

Delphi Unicode – Entendendo os avisos (warning) do compilar sua aplicação
http://www.andreanolanusse.com/blogpt/delphi-unicode-entendo-os-avisos-warning-do-compilar-sua-aplicacao/

Unicode Statistics Tool
http://cc.embarcadero.com/item/27398

Entendendo Unicode para migrar sua aplicação Delphi
http://www.andreanolanusse.com/blogpt/entendendo-unicode-para-migrar-sua-aplicacao-delphi/

Delphi in a Unicode World Part I: What is Unicode, Why do you need it, and How do you work with it in Delphi?
http://edn.embarcadero.com/article/38437

Delphi in a Unicode World Part II: New RTL Features and Classes to Support Unicode
http://edn.embarcadero.com/article/38498

Delphi in a Unicode World Part III: Unicodifying Your Code
http://edn.embarcadero.com/article/38693

Code Page Identifiers
http://msdn.microsoft.com/en-us/library/dd317756(v=VS.85).aspx

STRING TYPES IN DELPHI
http://delphi.about.com/od/beginners/l/aa071800a.htm

Delphi 2009 Unicode
http://www.drbob42.com/examines/examinA7.htm

Delphi Unicode Migration for Mere Mortals: Stories and Advice from the Front Lines
http://www.embarcadero.com/images/dm/technical-papers/delphi-unicode-migration.pdf

A Brief History of Strings
http://www.codexterity.com/delphistrings.htm

Unihan Database Lookup
http://unicode.org/charts/unihan.html

Last Resort Font
http://unicode.org/policies/lastresortfont_eula.html

Unicode Character Search
http://www.fileformat.info/info/unicode/char/search.htm?q=2a6d6&han=Y&preview=entity

Blogs do autor
http://www.vitorrubio.blogspot.com/
http://twitter.com/vitorrubio

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **4000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

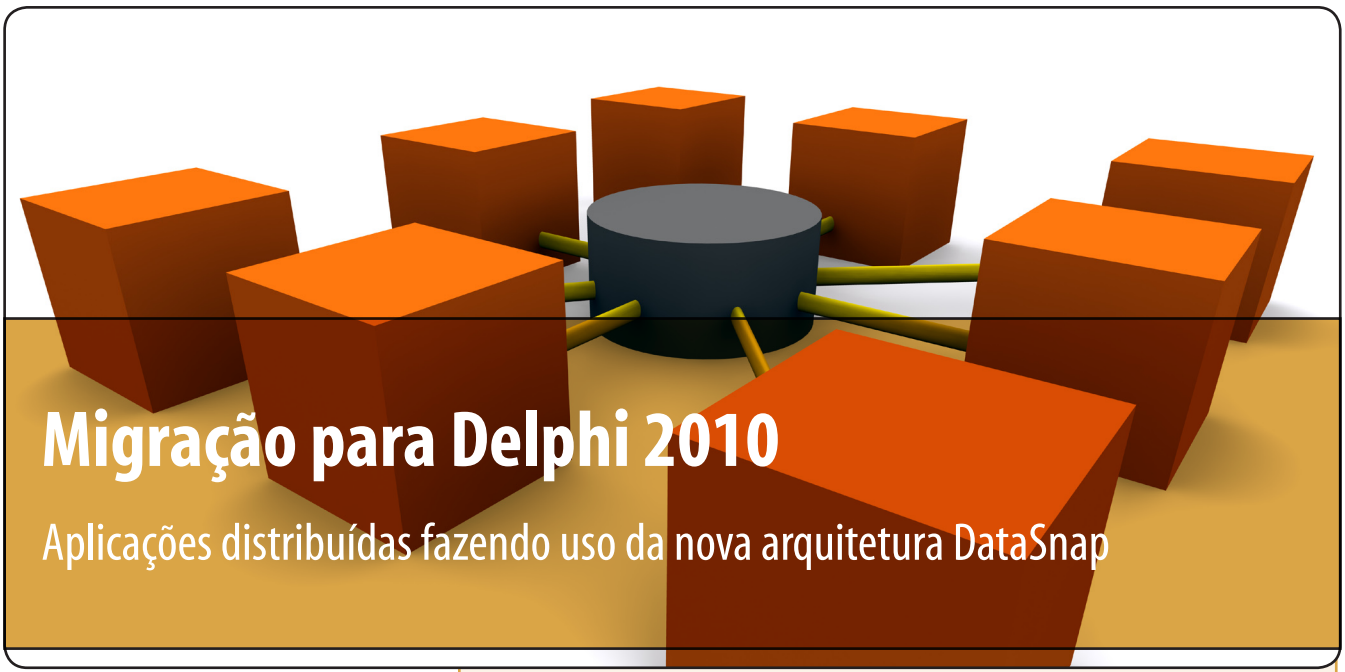
Todas as vídeos que você sempre quis ver a partir **R$ 0,99!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software



Migração para Delphi 2010

Aplicações distribuídas fazendo uso da nova arquitetura DataSnap

Do que trata o artigo

Mostrar de forma prática como migrar uma solução DataSnap tradicional desenvolvida em Delphi 7, que utiliza Web Services e SOAP para a comunicação entre Servidor de Aplicação e camada cliente, para a nova arquitetura DataSnap que acompanha a mais recente versão do Delphi, a 2010.

Para que serve

Ao atualizar a tecnologia vamos ganhar recursos novos e aprimorados, mantendo a aplicação pronta para os desafios atuais, também vamos entender as facilidades incluídas nesta nova versão. Ao empregar o uso do DataSnap teremos uma aplicação dinâmica, pois a lógica estará centralizada permitindo que vários clientes a acessem. Podemos ter clientes Desktop ou até mesmo páginas web consumindo os dados da aplicação.

Em que situação o tema é útil

Quando se precisa acessar um banco de dados a partir de qualquer lugar da Internet. Por exemplo, centralizar o banco de dados e servidor de aplicação na matriz da empresa e distribuir as aplicações clientes para as filiais, que terão acesso aos dados em tempo real e que se mantêm sempre atualizados, sem a necessidade de sincronização. Ainda é possível oferecer sua aplicação como um serviço: seus clientes podem pagar um taxa, por exemplo, e acessar a aplicação através da Internet, sem a necessidade de nenhuma configuração local na empresa, tudo fica configurado e armazenado no servidor.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Natural de Araçatuba-SP e formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador (Delphi e Java) há 4 anos, sendo 3 destes numa empresa do ramo de telecomunicações.

Resumo do DevMan

A nova tecnologia DataSnap para desenvolvimento multicamadas, lançada com o Delphi 2009 e aprimorada no Delphi 2010, é parcialmente baseada na arquitetura dbExpress e não mais em COM, que foi totalmente retirado da arquitetura. Dito isto será mostrada a migração, de forma prática, de um sistema desenvolvido com o DataSnap "antigo" para esta nova versão, mostrando tudo o que muda durante o desenvolvimento, como por exemplo componentes e recursos que serão descartados e outros que serão adicionados.

O desenvolvimento multicamadas no Delphi teve início com o MIDAS, ainda no Delphi 3. Esta tecnologia passou por constantes evoluções a cada nova versão, tanto que no Delphi 6, foi renomeada para DataSnap onde foi mantida até a versão 2007. Já com o lançamento do Delphi 2009 foi introduzida uma arquitetura totalmente nova, com a remoção da dependência COM e uma forma mais leve de produção dos lados cliente e servidor, tudo baseado na arquitetura do dbExpress do próprio Delphi. Essa versão, apesar de até então só oferecer suporte à conectividade TCP/IP, representou um marco na evolução da IDE, tanto que foi rebatizada como DataSnap 2009, para que houvesse uma distinção em relação às versões anteriores. Seguindo essa linha evolutiva, no ano passado com o lançamento do Delphi 2010, foi novamente apresentado um novo DataSnap, agora baseado em seu antecessor DataSnap 2009 e com mais melhorias e novidades como, por exemplo, o suporte a REST e JSON, conectividade utilizando o protocolo HTTP entre outras.

Imagine agora que você tenha um projeto DataSnap desenvolvido tempos atrás e que ainda o utiliza em produção até hoje. Você, ao acompanhar as mais recentes novidades do Delphi, seja na Web ou aqui mesmo nessa revista, percebeu todo o alarde feito sobre as melhorias do novo DataSnap e sentiu necessidade ou até mesmo curiosidade em atualizar tal aplicação para esta nova arquitetura.

Em vista disso, a proposta deste artigo é justamente mostrar a migração de uma aplicação desenvolvida com o DataSnap tradicional para a nova arquitetura DataSnap. Faremos isso de uma maneira bem prática, com foco exclusivo na migração propriamente dita, ou seja, de imediato construiremos a aplicação na arquitetura tradicional e logo em seguida a transportaremos para a nova arquitetura, mostrando todas as inclusões e exclusões de recursos entre um projeto e outro. Para valorizar ainda mais este processo, irei utilizar a premissa de um projeto que utiliza o DataSnap tradicional e SOAP ainda no Delphi 7, publicado tempos atrás em um artigo da própria ClubeDelphi, mais especificamente na edição 70 (ver seção links) de autoria do editor desta revista, Guinther Pauli. Uma observação a se fazer é que apesar do projeto citado utilizar o DataSnap tradicional, desenvolvido no Delphi 7, neste artigo utilizaremos o Delphi 2010 para todo o desenvolvimento prático. Dessa forma, comprovaremos a retro compatibilidade da IDE, mostrando que tudo que era feito anteriormente no desenvolvimento DataSnap ainda pode ser feito na nova versão. Para um melhor entendimento a partir de agora trataremos como "tradicional", a primeira versão da aplicação que desenvolveremos.

Mudanças no DataSnap

Como estamos falando da migração de uma aplicação DataSnap, da tradicional para a nova arquitetura, é importante que você conheça a diferença essencial entre ambas:

- Para o desenvolvimento do Servidor não existe mais envolvimento com COM. Agora os aplicativos cliente e servidor se comunicam diretamente sobre o protocolo TCP/IP ou HTTP. Não há mais a necessidade de se registrar o Servidor de Aplicação no Sistema Operacional;
- Na arquitetura tradicional, fazendo uso de COM, o Servidor era iniciado conforme fosse solicitado. Já no novo modelo você deve executar manualmente o Servidor de Aplicação ou criar um serviço para este;
- A partir da versão 2009 o Delphi trouxe novos componentes específicos para o desenvolvimento DataSnap, proporcionando ao desenvolvedor uma menor escrita de código;
- TDSServerModule é uma nova classe também incorporada na versão 2009, que herda de TDataModule e também serve como um container. A novidade é que esta já vem com a diretiva de compilação $MethodInfo habilitada, que permite ao DataSnap obter informações da classe do servidor através da RTTI;
- Não é necessário fazer o deploy de nenhuma DLL no lado cliente. Toda conectividade entre as partes cliente e servidor se dá pelo novo driver dbExpress DataSnap.

Visão geral da solução tradicional

Trata-se de uma aplicação multicamadas (três camadas) DataSnap, que fará uso de SOAP e WebServices, além do desenvolvimento Desktop (VCL Win32) do Delphi para a construção de uma solução distribuída para Internet. Para a parte de banco de dados será usado o SGBD Firebird 2.1 (ver seção Links) e mais precisamente sua tabela Employees. O Web Service será o meio para a comunicação entre o Servidor de Aplicação e a aplicação cliente utilizando para tal o protocolo SOAP (Simple Object Access Protocol), trafegando dados no formato XML. Em outras palavras, esta solução permitirá que as aplicações clientes estejam distribuídas geograficamente, uma vez que terão acesso ao Servidor de Aplicação e consequentemente às informações do banco de dados via Web. Este tipo de projeto que criaremos para o Servidor de Aplicação irá gerar uma dll que deverá ser publicada em um servidor Web IIS (Internet Information Services) do Windows, portanto, é imprescindível que você o tenha instalado e configurado em sua máquina. Por se tratar de um procedimento básico e muito peculiar de acordo com o ambiente em que se está trabalhando, não fará parte do escopo deste artigo.

Nota do DevMan

SOAP, acrônimo de Simple Object Access Protocol, é um protocolo que utiliza tecnologias baseadas em XML (texto) para troca de dados estruturados entre plataformas distribuídas. Essa característica torna seu uso muito comum em ambientes que utilizam servidores HTTP.

Nota do DevMan

Microsoft COM, acrônimo de Component Object Model, é uma tecnologia da família dos sistemas operacionais Windows que permite que componentes de software se comuniquem. No Delphi, é utilizado para viabilizar a comunicação em sistemas distribuídos, entre as partes cliente e servidora, fazendo uso de interfaces.

Através da aplicação cliente será possível buscar um registro da tabela de Empregados (Employees) do banco de dados, para que se possa visualizar estes dados e, caso necessário, alterá-los. Uma observação a se fazer é que esta tabela possui um campo denominado DEPT_NO, que nada mais é que uma referência a outra tabela (Department) do banco de dados. Mostraremos essa informação utilizando um campo do tipo Lookup, que mostra o nome do departamento de acordo com sua referência gravada na tabela Employees. Além disso, criaremos um método no Servidor de Aplicação cujo objetivo é obter o Total de Vendas de um determinado empregado. Este procedimento deverá ser consumido pela aplicação cliente, lembrando que a aplicação cliente não possui acesso direto à camada de dados (banco de dados) e todas as requisições serão realizadas via Servidor de Aplicação.

Para um maior entendimento da proposta do projeto do artigo que será usado na migração, veja o esboço de sua arquitetura (**Figura 1**) que foi publicado originalmente na edição 70.

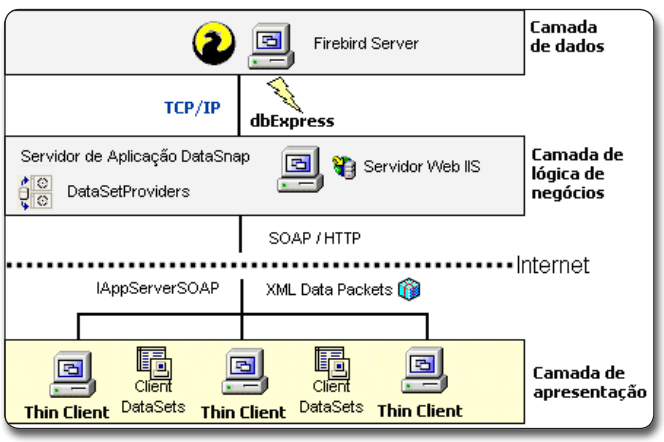


Figura 1. Visão da arquitetura do projeto original

Na figura você percebe claramente a divisão das camadas:

• Camada de dados: onde ficará o banco de dados que, neste caso, será o SGBD Firebird 2.1;

• Camada de lógica de negócios: é o que chamaremos de Servidor de Aplicação, onde ficarão centralizadas as regras de negócio da aplicação e acesso a dados. Contém componentes dbExpress para acesso ao banco de dados;

• Camada de apresentação: são as aplicações clientes, conhecidas como Thin Clients (clientes leves), que serão executáveis bem leves, já que todo o código das regras de negócios estará centralizado no Servidor de Aplicação. Nesta camada estarão os ClientDataSets.

Requisitos e preparação do ambiente

Para o desenvolvimento deste exemplo você deverá ter instalado em sua máquina o Delphi 2010 e o SGBD Firebird 2.1. Estes itens podem ser baixados gratuitamente diretamente do site da Embarcadero, e do site oficial do Firebird respectivamente (ver seção links no final do artigo). A parte da instalação de tais ferramentas não será abordada, visto que todas são bem intuitivas. Como dica, a critério de organização, crie uma pasta chamada DataSnap em um diretório qualquer e, dentro desta, outras duas pastas: Server e Client. Essas pastas serão os locais onde salvaremos os arquivos do projeto deste artigo.

Aplicação tradicional - Servidor de Aplicação

Dando início à parte prática do projeto, iremos começar pelo Servidor de Aplicação. Como já foi mencionado anteriormente, iremos construí-lo utilizando a arquitetura tradicional do tipo SOAP Server. Portanto, como o IDE aberto acesse o menu File > New > Other > WebServices > SOAP Server Application (**Figura 2**).

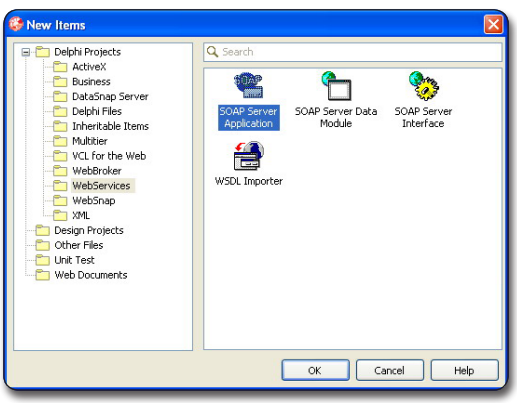


Figura 2. Opção para criação de um servidor SOAP

Selecione o item e confirme no botão OK. A seguir, o Delphi exibirá uma mensagem perguntando se você deseja criar uma interface SOAP, apenas escolha Não, pois está fora do escopo deste artigo. A partir deste momento o Delphi já automaticamente cria um projeto contendo um TWebModule (**Figura 3**), que é responsável por permitir que o Servidor de Aplicação possa responder às requisições HTTP vindas da camada cliente, direcionando estes pedidos e suas respostas para uma ação apropriada. Ele serve também como um container para alguns componentes não visuais que já vieram inseridos no Web Module no momento da criação do projeto. Apenas para não passar em branco, veja uma rápida descrição de tais componentes:

• HTTPSoapDispatcher: numa solicitação SOAP, este componente é quem a encaminha para um componente que interprete tal mensagem, neste caso o HTTPSoapPascalInvoker;

• HTTPSoapPascalInvoker: ao receber uma mensagem SOAP, este componente a direciona para uma interface que seja capaz de processar a solicitação;

• WSDLHTMLPublish: gera uma página HTML que publica a lista de documentos WSDL (Web Service Definition Language) que representam a aplicação Web Service.

Salve tudo (Shift+Ctrl+S) na pasta Server nomeando o projeto como "Server" e a unit do WebModule como uWM. Para este

último, aproveite para mudar sua propriedade Name para WM. Como já foi dito, este tipo de projeto irá gerar uma dll a ser publicada no IIS. Portanto iremos configurar a saída do projeto diretamente para a pasta padrão do IIS. Para tal, acesse o menu Project > Options > Delphi Compiler e em Output directory digite C:\Inetpub\Scripts\.

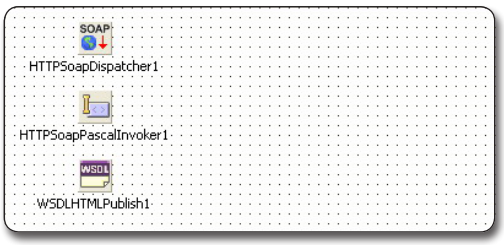


Figura 3. WebModule criado

Dando continuidade ao exemplo, iremos agora configurar a parte de acesso a dados que ficará a cargo do Servidor de Aplicação. O primeiro passo é configurar uma conexão ao banco Employee do Firebird. Portanto, vá até a guia Data Explorer e clique com o botão direito sobre o nó Firebird e selecione Add New Connection. Lembrando que estamos fazendo dessa forma, pois na versão atual do dbExpress não mais criamos ou configuramos uma conexão dando um duplo-clique sobre o SQLConnection. Dê o nome de EMPLOYEE para a nova conexão e configure seus parâmetros conforme a **Figura 4**. Veja que em Database Name indicamos o caminho padrão do banco Employee que acompanha a instalação do Firebird e em User Name e Password, apenas informamos o usuário e senha padrão respectivamente: SYSDBA e masterkey. Por fim, teste a conexão e se tudo estiver perfeito clique em OK.

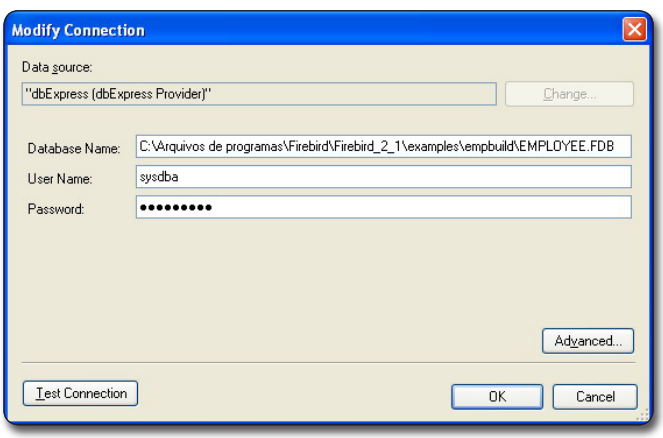
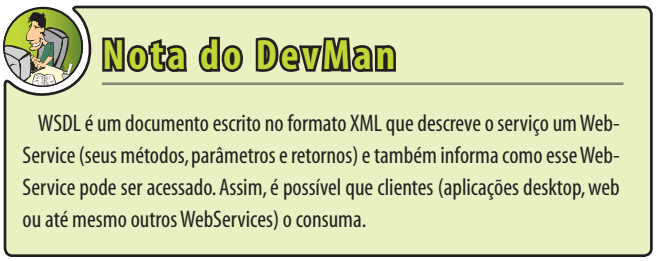


Figura 4. Configurando uma nova conexão com o banco Employee

Definida a conexão, precisamos agora de um container, uma espécie de Data Module para mantermos os componentes de acesso a dados. Para o tipo de aplicação que estamos desenvolvendo existe um item específico para esse propósito chamado SOAP Data Module. Para inserir um objeto deste tipo ao projeto acesse o menu File > New > Other> WebServices e selecione o item SOAP Server Data Module, conforme mostra a **Figura 5**.



Nota do DevMan

WSDL é um documento escrito no formato XML que descreve o serviço um Web-Service (seus métodos, parâmetros e retornos) e também informa como esse Web-Service pode ser acessado. Assim, é possível que clientes (aplicações desktop, web ou até mesmo outros WebServices) o consuma.

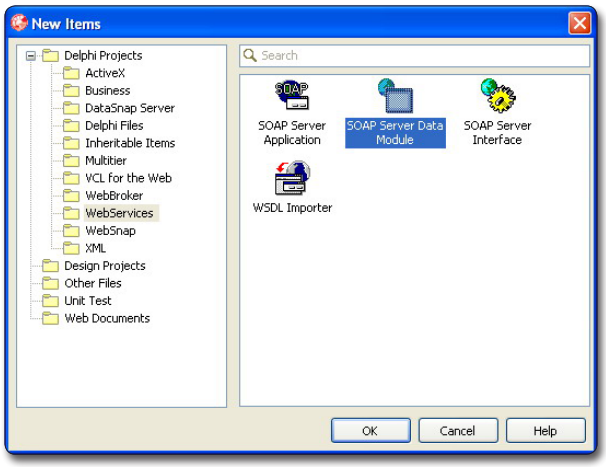


Figura 5. Escolhendo a opção para inserir um SOAP Server Data Module

Ao selecioná-lo é apresentada uma caixa de diálogo para que seja inserido o Module Name. Digite SoapDM e confirme. Feito isso o Delphi já o adiciona ao projeto. Salve sua unit como uSoapDM. Agora basta arrastar a conexão definida no Data Explorer para dentro dele, dessa forma já temos um componente TSQLConnection devidamente configurado. A seguir, da aba dbExpress, insira dois componentes TSQLDataSet (sdtEmployee e sdtDepartment) e da aba DataAccess mais três TDataSetProviders (dspEmployee, dspDepartment e dspTotalVendasEmpregado). Para os dois primeiros, conecte-os ao SQLConnection através da propriedade de mesmo nome e vincule sua propriedade DataSet a cada um dos TSQLDataSets. O último DataSetProvider será abordado mais adiante.

Falta agora definirmos as instruções SQL de cada componente SQLDataSet. Portanto, para os componentes sdtEmployee e sdtDepartment, respectivamente, defina a SQL em sua propriedade CommandText conforme a **Listagem 1**.

Conforme pode ser visto na listagem apresentada, a primeira instrução SQL, referente à tabela Employees, é parametrizada. É necessário configurarmos esse parâmetro no próprio componente. Portanto no sdtEmployee vá em sua propriedade Params e configure o parâmetro como ftSmallint e ptInput. Ainda neste componente, acesse sua lista de Fields com um duplo-clique e pressione Ctrl + F para que todos os Fields sejam adicionados na lista. Para cada Field, em sua propriedade ProviderFlags deixe como True somente o item pfInUpdate, exceto para o Field Emp_No que, por ser tratar da chave primária da tabela, terá marcado como True os itens pfInUpdate, pfInWhere e pfInKey. Para o sdtDepartment não será necessária nenhuma alteração uma vez que não estaremos fazendo nenhuma ação direta sobre ele.

Listagem 1. Instruções SQL dos componentes SQLDataSet

```
--sdtEmployee--
select EMP_NO, FIRST_NAME, LAST_NAME, DEPT_NO from EMPLOYEE
where EMP_NO =:EMP_NO

--sdtDepartment--
select DEPT_NO, DEPARTMENT from DEPARTMENT
```

Passando agora para o componente dspEmployee, altere sua propriedade UpdateMode para upWhereKeyOnly para que a cláusula Where da instrução SQL de atualização seja montada com base no campo que definimos como pfInKey. Em Options, marque as opções poIncFieldsProps e poPropagateChanges para que todas as configurações que fizemos no sdtEmployee sejam repassadas à camada cliente, mais precisamente ao ClientDataSet que iremos utilizar na camada cliente.

Por fim, para ver se o Servidor de Aplicação está funcionando corretamente, dê um Build no projeto (Shift + F9) para que a dll seja gerada na pasta do IIS. A seguir abra o browser de sua máquina e digite na barra de endereços a seguinte URL: *http://localhost/scripts/Server.dll/soap/ISoapDM*. Caso tudo esteja de acordo, você verá uma página HTML simples, que nada mais é que uma página gerada pelo componente WSDLHTMLPublish presente no Web Module, que lista todas interfaces registradas no nosso servidor, inclusive a ISoapDM (**Figura 6**) que será usada para troca de dados via HTTP.

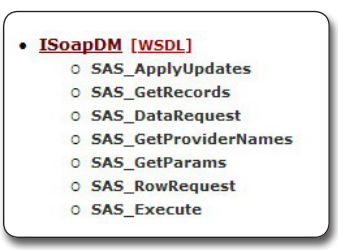


Figura 6. Trecho da página gerada mostrando a interface ISoapDM

Método de Servidor

Iremos agora criar um método no Servidor de Aplicação que futuramente possa ser consumido pela aplicação cliente. Tal método, conforme citado no começo do artigo, irá nos retornar o total de vendas de um determinado empregado. Portanto no código do SOAP Data Module, declare na seção public o método TotalVendasEmpregado conforme a seguir:

```
function TotalVendasEmpregado(EmpNo: Smallint): Integer;
```

Logo depois, sua implementação deve ficar igual à **Listagem 2**.

Na **Listagem 2** inicialmente declaramos uma variável do tipo TSQLQuery que será utilizada para se fazer uma consulta a uma tabela do banco de dados que retornará o somatório do total de vendas de um determinado empregado passado como parâmetro. A seguir este objeto é instanciado e a consulta é realizada. Logo, o Result da função será o próprio retorno desta consulta. Finalmente, após seu uso, o objeto é liberado da memória. Agora, para podermos disponibilizar este método para a camada cliente utilizaremos o componente dspTotalVendasEmpregado que até então não foi configurado. Na verdade, o que faremos é somente codificar seu evento OnDataRequest. É através deste evento que a camada cliente, através de um ClientDataSet, irá se comunicar com este Provider. Portanto no evento citado, insira o código a seguir:

```
function TSoapDM.dspTotalVendasEmpregadoDataRequest(Sender: TObject;
 Input: OleVariant): OleVariant;
begin
 Result:= TotalVendasEmpregado(Input);
end;
```

Aqui apenas atribuímos ao Result do evento, que será passado à camada cliente, o resultado da função TotalVendasEmpregado. E para chamar esta função, passamos como parâmetro para ela o parâmetro Input do próprio evento, cujo valor será atribuído na camada cliente através do método DataRequest de um ClientDataSet.

Listagem 2. Corpo do método TotalVendasEmpregado

```
function TSoapDM.TotalVendasEmpregado(EmpNo: SmallInt): Integer;
var
 Qry: TSQLQuery;
begin
 try
  Qry:= TSQLQuery.Create(nil);
  Qry.SQLConnection:= EMPLOYEE;
  Qry.SQL.Add('select sum(TOTAL_VALUE) from SALES');
  Qry.SQL.Add('where SALES_REP = :EMP_NO');
  Qry.ParamByName('EMP_NO').AsInteger:= EmpNo;
  Qry.Open;
  Result:= Qry.Fields[0].AsInteger;
 finally
  FreeAndNil(Qry);
 end;
end;
```

Aplicação tradicional - Aplicação Cliente

Partiremos para a construção da aplicação cliente de nosso exemplo. Para tal, adicione no Project Manager um novo projeto VCL Forms Application. Salve-o na pasta Client como ClientWin32 e a unit do Form como uClientForm. Mude o Caption do Form para Client – Win32. A seguir salve também o Project Group como DataSnapSolution na pasta raiz que nomeamos como DataSnap. O próximo passo, por questões de organização, é inserir um DataModule ao projeto (File > New > Others > Delphi Files > DataModule) onde concentraremos os TClientDataSets além do componente de conexão ao Servidor de Aplicação. Salve sua unit como uDM e lhe dê o nome de DM. Insira nele um componente TSoapConnection, presente na paleta WebServices (**Figura 7**), que fará a conexão com o Servidor de Aplicação usando o protocolo SOAP.

Imediatamente, digite em sua propriedade URL o caminho (path) completo da interface do SOAP Data Module do Servidor

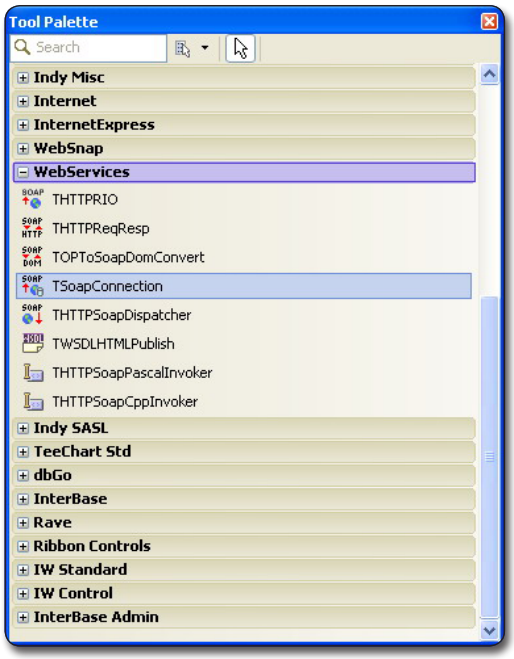


Figura 7. Selecionando o componente TSoapConnection

de Aplicação, que neste caso será: *http://localhost/scripts/Server.dll/soap/ISoapDM* e logo após mude sua propriedade Connected para True. Adicione agora dois componentes TClientDataSet da paleta DataAccess, nomeando-os respectivamente como cdsEmployee e cdsDepartment, que armazenarão localmente os dados vindos do servidor. Para cada um destes, aponte a propriedade RemoteServer para o SoapConnection1. Agora precisamos ligá-los aos DataSetProviders localizados no Servidor de Aplicação. Para isso, acesse sua propriedade ProviderName e repare que a IDE já apresenta para você todos os Providers disponíveis (**Figura 8**) do servidor, portanto, para o cdsEmployee aponte o dspEmployee e para o cdsDepartment o dspDepartment. Ainda nos ClientDataSets, para cada um, acesse sua lista de Fields e adicione todos (Ctrl+F) a ela. Neste momento não precisamos fazer nenhum tipo de ajuste nestes Fields adicionados, uma vez que já definimos no componente DataSetProvider do servidor para que seja repassada as configurações definidas naquela camada. A única exceção é para a criação do campo Lookup, citado no início do artigo, que criaremos no cdsEmployee. Portanto, neste componente, novamente em sua lista de Fields crie um novo Field (Ctrl+N) do tipo Lookup, conforme mostra a **Figura 9**. Altere todas ProvidersFlags deste novo campo para False. Como último ajuste para este componente, clique com o botão direito sobre ele e no menu que se abre escolha a opção FetchParams, para que o parâmetro definido no Servidor de Aplicação seja repassado à camada cliente.

Partindo agora para a parte visual da aplicação cliente, decore o formulário fazendo-o ficar semelhante à **Figura 10**.

O visual é bem simples e foram usados os respectivos componentes:

- 1 TDataSource, ligado ao ClientDataSet cdsEmployee, presente no DataModule;
- 4 TLabels, cujos Captions representam cada um dos Fields do cdsEmployee;
- 3 TDBEdits, ligados ao DataSource e apontados para cada um dos seus respectivos Fields;
- 1 TDBLookupComboBox, ligados ao DataSource e vinculado ao campo Lookup criado;
- 1 TEdit (edtBuscar);
- 3 TButtons (btnApply, btnTotalVendas e btnBuscar).

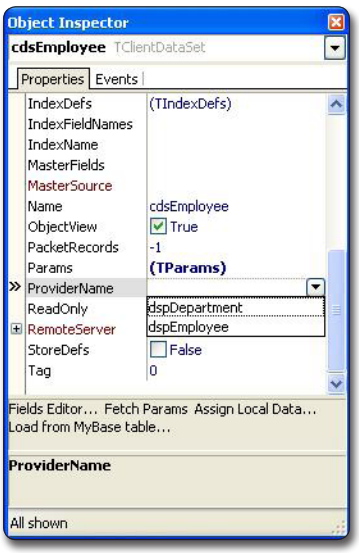


Figura 8. Lista de Providers disponíveis para a camada cliente

Figura 9. Criando campo do tipo Lookup

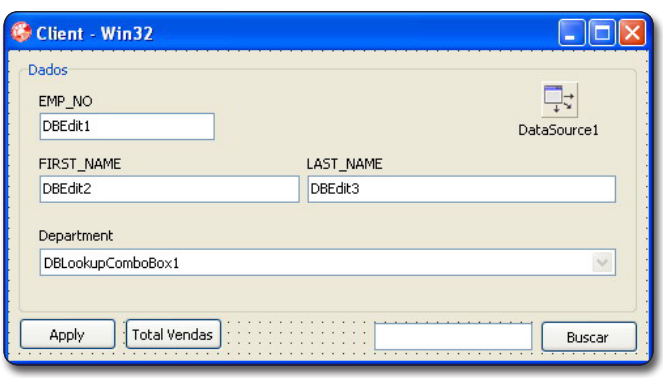


Figura 10. Visual da aplicação cliente

Definido o visual, resta a implementação do código desta tela. Logo de início, para termos acesso aos ClientDataSets no formulário é necessário referenciar na seção uses (Alt+F11) o DataModule DM. Feito isso, primeiramente iremos codificar o evento OnClick do botão btnBuscar. Portanto, insira nele o código visto a seguir:

```
procedure TForm1.btnBuscarClick(Sender: TObject);
begin
 DM.cdsEmployee.Close;
 DM.cdsEmployee.Params[0].AsInteger:= StrToInt(edtBuscar.Text);
 DM.cdsEmployee.Open;
end;
```

Neste código estamos manipulando o ClientDataSet relativo à tabela Employee. Primeiramente o fechamos, através de seu método Close, para que possamos passar um valor ao parâmetro a ser utilizado na consulta de dados no servidor. Após, o ClientDataSet é aberto através de seu método Open, já disponibilizando para esta aplicação os dados enviados pelo Servidor de Aplicação, caso houverem. Para os outros dois botões restantes, também para seu evento OnClick, insira o código da **Listagem 3.**

Listagem 3. Códigos do evento OnClick dos botões btnApply e btnTotalVendas

```
procedure TForm1.btnApplyClick(Sender: TObject);
begin
 DM.cdsEmployee.ApplyUpdates(0);
end;

procedure TForm1.btnTotalVendasClick(Sender: TObject);
begin
 if DBEdit1.Text <> '' then
   ShowMessage('O Total de Vendas deste Empregado é: ' +
     IntToStr(DM.cdsTotalVendasEmpregado.DataRequest(DBEdit1.Text)));
end;
```

Nesta listagem, para o botão btnApply apenas chamamos o método ApplyUpdates do próprio ClientDataSet que está sendo manipulado, para que as alterações realizadas em tela sejam enviadas ao Servidor de Aplicação e efetivamente salvas no banco de dados. Já para o botão btnTotalVendas é feita uma verificação se há algum registro sendo visualizado naquele momento, através do preenchimento do controle DBEdit1. Caso haja, é mostrada uma mensagem com o total de vendas do referido empregado, referente ao resultado do método disponibilizado no Servidor, passando o parâmetro desejado via DataRequest do próprio ClientDataSet. Para testar, dê um Build no projeto (Shift+F9) e execute a aplicação (F9). Busque algum registro e realize alguma mudança em seus dados. Refaça a pesquisa e veja que os dados foram realmente persistidos na tabela do banco de dados.

Início da Migração - Servidor de aplicação

Daremos início agora à migração propriamente dita, a começar pelo projeto Server. A primeira coisa a se fazer nele é inserir um container da classe TDSServerModule, que herda de TDataModule, incluindo a interface IAppServer (ver nota). Ele substituirá o atual SoapDataModule, uma vez que, no cenário do novo DataSnap o TDSServerModule tem a função de concentrar os métodos e objetos do servidor que estarão disponíveis às aplicações clientes, tanto que ao se usar o wizard oferecido pelo Delphi para criar aplicações multicamadas, ele é criado com um nome bem intuitivo: ServerMethods. Portanto, mova todos os componentes e métodos do SoapDataModule para ele, exceto o Provider dspTotalVendasEmpregado, que estava provendo à camada cliente o acesso ao método definido no servidor e será descartado neste momento. Este detalhe será explicado mais adiante.

Voltando ao desenvolvimento, com o projeto Server ativo, adicione um TDSServerModule acessando File > New > Other > DataSnap Server > Server Module, conforme mostra a **Figura 11**. Salve sua unit como uSM e lhe dê o nome de SM.

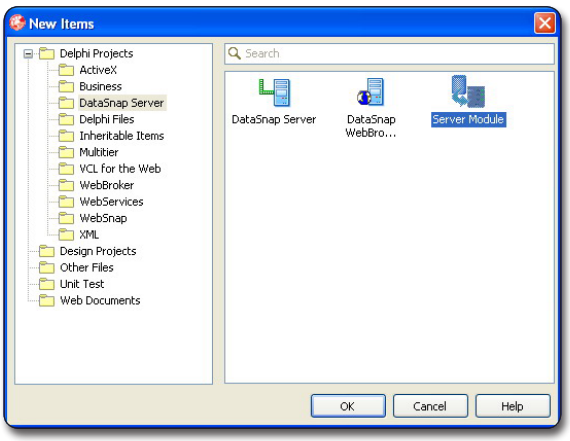


Figura 11. Adicionando um ServerModule ao projeto

Ao adicionarmos este Server Module, já podemos descartar o SoapDataModule, portanto no Project Manager dê um clique com o botão direito do mouse sobre ele e selecione a opção Remove From Project. A seguir, precisamos modificar o WebModule de forma a descartar o uso de seus componentes atuais, uma vez que estes são relativos ao SOAP, pertencem à paleta Web Services e, logo, serão substituídos pelos novos componentes DataSnap presentes na nova paleta DataSnap Server. Estes proporcionarão agora toda configuração e conectividade necessária. Portanto, remova todos os componentes presentes no WebModule e logo após, navegue até a paleta DataSnap Server e arraste os seguintes componentes para o container, como mostra a **Figura 12**:

- TDSServer: componente central de um servidor DataSnap. Gerencia a parte de transporte e as classes de servidor;
- TDSServerClass: componente que funciona como uma fábrica de objetos de cada classe que se deseja expor aos clientes;
- TDSHTTPWebDispatcher: responsável por interceptar as requisições HTTP ao servidor, numa arquitetura DataSnap.

Nota do DevMan

A interface IAppServer surgiu na época do Midas 3, mais especificamente no Delphi 5, disponibilizando à arquitetura DataSnap métodos necessários para a comunicação entre cliente e servidor em aplicações de DataSets distribuídos.

Para os componentes DSHTTPWebDispatcher e DSServer Class, ligue-os ao DSServer1 através de sua propriedade Server. Agora precisamos codificar alguns eventos para que nossa estrutura DataSnap fique corretamente configurada. Primeiramente, no próprio WebModule, para fins de execução de testes futuros, codificaremos sua ação padrão de resposta HTTP, conforme código a seguir:

```
procedure TWM.WMDefaultHandlerAction(Sender: TObject;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
 Response.Content := '<html><heading/><body>DataSnap Server funcionando
 </body></html>';
end;
```

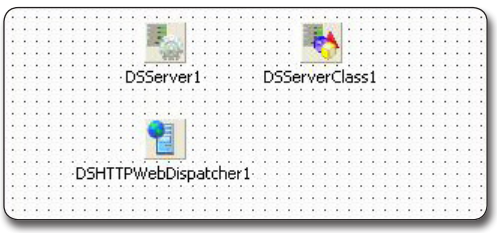


Figura 12. Novos componentes DataSnap adicionados

Conforme pode ser visto no código apresentado, quando houver alguma solicitação ao WebModule, sua ação padrão irá retornar um estrutura HTML simples com os dizeres "DataSnap Server funcionando", indicando que o Servidor de Aplicação está trabalhando adequadamente. Na verdade este método não é essencial ao projeto, porém, será muito útil para testarmos o funcionamento do servidor de aplicação quando publicarmos a dll do projeto no servidor Web.

A próxima codificação a ser feita é no evento OnGetClass, que é o método que especifica a classe do servidor que será exposta, uma vez que, todos os seus métodos públicos poderão ser chamados por qualquer cliente. Portanto no evento citado, insira o código a seguir:

```
procedure TWM.DSServerClass1GetClass(DSServerClass: TDSServerClass;
var PersistentClass: TPersistentClass);
begin
 PersistentClass := uSM.TSM; //dar uses em uSM
end;
```

Neste código, PersistentClass recebe o nome da classe do servidor que será exposta. Feito isso, o Servidor de Aplicação está pronto, basta dar um build (Shift + F9) para que a dll do projeto seja gerada. Conforme configurado no projeto, ela será gerada em C:\Inetpub\Scripts. Mude conforme sua necessidade, mudando o path em Project > Options > Delphi Compiler > Outuput directory. Para podermos testar o funcionamento do servidor que acabamos de remodelar abra o navegador de internet e digite a seguinte URL: *http://localhost/Scripts/Server.dll*.

Com isso, fizemos uma solicitação ao servidor via HTTP, logo, o método DefaultHandlerAction foi executado e o resultado obtido você pode ver conforme mostra a **Figura 13**.

Ainda no código do WebModule, algumas referências não são mais necessárias, como por exemplo na seção uses, as referências às units relativas a Soap e Web Services, como por exemplo: InvokeRegistry, WSDLIntf, WebServExp, WSDLBind, WSDLPub, SOAPPasInv, SOAPHTTPPasInv, SOAPHTTPDisp e WebBrokerSOAP. Todas podem ser removidas da seção.

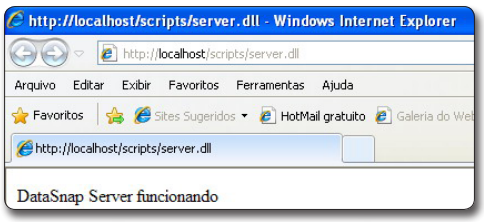


Figura 13. Testando o servidor de aplicação

Aplicação cliente

Feitas as alterações necessárias no Servidor de Aplicação, iremos agora adequar a parte cliente do grupo de projetos do exemplo. Uma teoria a ser reforçada neste momento é de que a partir do DataSnap 2009, a conectividade entre cliente e servidor é de responsabilidade do novo driver dbExpress DataSnap. Portanto, novamente no Data Explorer, veja que existe um item DataSnap exclusivo para este tipo de conexão. Criaremos então uma nova conexão, assim como foi feita com o Firebird. Para isso dê um clique com o botão direito no nó DATASNAP, e no menu que se abre escolha Add New Connection. Nomeie a conexão como SERVER. Configure os parâmetros da conexão conforme a **Figura 14**.

Conforme visto na **Figura 14**, os seguintes parâmetros foram configurados:

- Protocol: indica o tipo de protocolo a ser usado na conexão. As opções são TCP/IP e HTTP. Para o nosso caso selecione este último;
- Host: é o host do servidor que, neste exemplo, trata-se de um servidor Web local, onde ficará a dll;
- Port: é a porta a ser usada pela conexão. Como não configuramos nenhuma em específico, deixamos o valor padrão 80, mas nada impede de você mudá-lo conforme seu ambiente;
- Path: caminho de onde se encontra o Servidor de Aplicação. Neste caso, indicamos o local onde publicamos a dll.

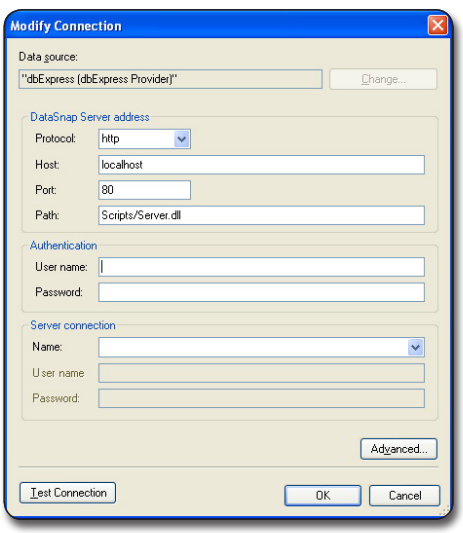


Figura 14. Configurando a conexão DataSnap

Por fim teste a conexão através do botão Test Connection. Caso esteja tudo certo, uma mensagem de sucesso será mostrada. A seguir, arraste a conexão criada para o DataModule. Um novo SQLConnection é adicionado, já totalmente configurado e que ficará encarregado de estabelecer uma conexão com o servidor, portanto, já podemos excluir o componente SoapConnection1. Veja que agora os ClientDataSets ficaram desprovidos de um Provider. Para resolvermos este problema, no novo DataSnap entra em cena um outro componente intermediário entre o SQLConnection e o ClientDataSet chamado DSProviderConnection e está presente sugestivamente na paleta DataSnap Client. Através dele poderemos enxergar os Providers do servidor. Portanto, insira dois destes componentes (dscEmployee e dscDepartment) já ligando-os ao SQLConnection SERVER. Preencha também sua propriedade ServerClassName com o nome da classe exposta no servidor, que neste caso deverá ser TSM. A seguir, em cdsEmployee e cdsDepartment, aponte a propriedade RemoteServer para cada um destes DSProviderConnection e veja que os providers voltaram a aparecer novamente na listagem da propriedade ProviderName. Terminando, por questões de funcionalidade, codifique os eventos OnCreate e OnDestroy do DataModue conforme mostra o código a seguir da **Listagem 4**. Na seção uses, declare a unit DSHTTPLayer, necessária para a comunicação via HTTP entre cliente e servidor.

Listagem 4. Codificação dos eventos OnCreate e OnDestroy do DataModule

```
procedure TDM.DataModuleCreate(Sender: TObject);
begin
  SERVER.Connected:= True;
end;

procedure TDM.DataModuleDestroy(Sender: TObject);
begin
  SERVER.Connected:= False;
end;
```

Nesta codificação estamos apenas certificando de que a conexão com o Servidor de Aplicação será estabelecida e desfeita, conforme o DataModule é criado e destruído da memória respectivamente. Se você notou, não citamos o cdsTotalVendasEmpregado.

Chamada a métodos do Servidor de Aplicação

Para completar a migração do projeto cliente, resta conseguirmos acessar o método TotalVendasEmpregados disponível no Servidor de Aplicação. Então como faremos para chamar este método remotamente nesta nova arquitetura DataSnap? No novo DataSnap as chamadas remotas de métodos do servidor são baseadas na RTTI do Delphi e para chamar efetivamente um método do Servidor de Aplicação, existem duas alternativas: utilizar o componente da paleta dbExpress, exclusivo para esse tipo de operação, SqlServerMethod ou através de uma classe Proxy.

Estaremos fazendo uso de ambas as opções. Portanto no DataModule da aplicação cliente, adicione um componente SqlServerMethod atribuindo a sua propriedade SQLConnection o componente de conexão SERVER. Feito isso, vá a sua propriedade ServerMethodName e veja que é apresentada uma lista de métodos do servidor que estão disponíveis, inclusive os métodos padrões da interface IAppServer além do método TotalVendasEmpregado, conforme pode ser visto na **Figura 15**.

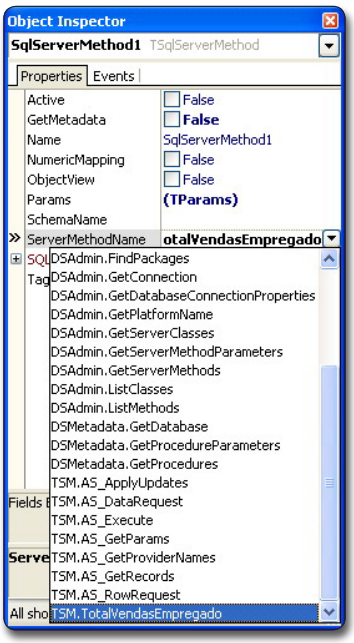


Figura 15. Listagem de métodos do servidor disponíveis

Selecione o método citado e a seguir clique em sua propriedade Params. Veja que nela já estão mapeados os parâmetros do método escolhido (**Figura 16**), que no caso são dois: um de entrada (EmpNo) e outro de saída (ReturnParameter). Dessa forma, você pode usar o componente da mesma forma que um componente StoredProcedure.

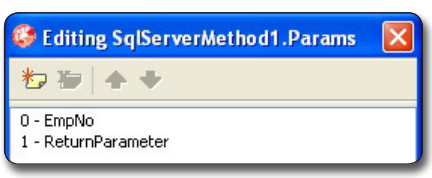


Figura 16. Lista de parâmetros do método do servidor

Para usarmos o método na aplicação cliente, no evento OnClick do botão btnTotalVendas, insira o código da **Listagem 5**.

Neste código fazemos uma chamada ao método definido no Servidor de Aplicação através do componente SqlServerMethod. O processo é simples: chamamos o método Close do componente para em seguida passar o valor do parâmetro de entrada, que será o valor definido no controle DBEdit1. Por conseguinte, é chamado o método ExecuteMethod do componente para que o método do servidor seja executado. Feito isso já temos no parâmetro de saída o valor do resultado do método e que é exibido em uma mensagem simples.

Listagem 5. Código do evento OnClick do botão btnTotalVendas

```
procedure TForm1.btnTotalVendasClick(Sender: TObject);
begin
 if DBEdit1.Text <> '' then
 begin
  DM.SqlServerMethod1.Close;
  DM.SqlServerMethod1.Params[0].AsInteger:= StrToInt(DBEdit1.Text);
  DM.SqlServerMethod1.ExecuteMethod;
  ShowMessage('O Total de Vendas deste Empregado é: ' +
  DM.SqlServerMethod1.Params[1].AsString;
 end;
end;
```

Iremos agora fazer uso da segunda abordagem mencionada anteriormente, criando uma classe proxy local na aplicação cliente. Para isso, vá ao componente SQLConnection SERVER e com o botão direito do mouse acesse a opção Generate Data Snap client classes (**Figura 17**). Neste momento o Delphi irá gerar automaticamente uma nova unit contendo a classe Proxy já com todos os métodos do servidor devidamente mapeados. Salve esta unit como uClientProxy.

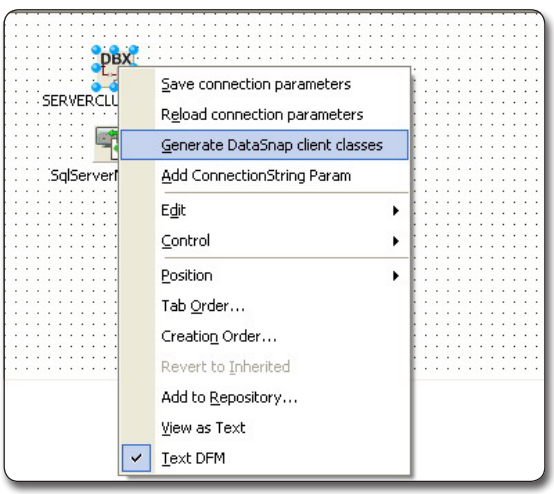


Figura 17. Opção para gerar classe Proxy na parte cliente

Para fazer uso desta classe adicione um novo TButton no formulário (btnTotalVendasProxy) e insira o código da **Listagem 6** em seu evento OnClick, lembrando que este código fará exatamente a mesma coisa que o código anterior usando o componente SqlServerMethod.

Listagem 6. Código do evento OnClick do botão btnTotalVendasProxy

```
procedure TForm1.btnTotalVendasProxyClick(Sender: TObject);
var
 ClientProxy: TSMClient; //dar uses em uClientProxy
begin
 try
  TSMClient.Create(DM.SERVER.DBXConnection);
  ShowMessage('O Total de Vendas deste Empregado é: ' +
  IntToStr(ClientProxy.TotalVendasEmpregado(StrToInt(DBEdit1.Text))));
 finally
  FreeAndNil(ClientProxy);
 end;
end;
```

Nota do DevMan

Uma classe proxy é a implementação do padrão de projeto Proxy, que tem a finalidade de interligar partes, como por exemplo uma aplicação servidora a uma aplicação cliente. Na aplicação cliente temos os métodos e propriedades do proxy, que durante o uso é repassado à classe/aplicação que o mesmo representa, formando assim uma espécie de delimitador lógico.

Neste código inicialmente foram declaradas duas variáveis: total, que irá guardar o valor de retorno do método do servidor e ClientProxy que serve para mantermos uma instância da classe proxy. A seguir instanciamos o objeto do tipo da classe Proxy, passando como parâmetro para o seu método Create o DBXConnection do nosso SQLConnection SERVER. Logo após, é chamado o método TotalVendasEmpregado, sendo o texto do DBEdit1 o valor de seu parâmetro de entrada. O resultado deste é atribuído à variável Total que é mostrada no texto da mensagem. Por fim, o objeto que instanciamos é liberado da memória.

Conclusão

Com base no que foi apresentado neste artigo você pode facilmente migrar suas aplicações multicamadas desenvolvidas em versões anteriores do IDE para o novo DataSnap, presente no Delphi 2010. Os ganhos são inúmeros, como o desenvolvimento mais facilitado, a ausência de dependência COM para comunicação entre as partes cliente e servidor além dos novos componentes que fornecem toda agilidade RAD. Agora você pode deixar suas aplicações com as mesmas funcionalidades de antes, porém, utilizando o que há de mais novo no desenvolvimento Delphi, utilizando esta, que já está sendo considerada pela comunidade como a melhor versão de todos os tempos. Lembrando que os fontes de ambas as versões da solução apresentada serão disponibilizados para download no Portal DevMedia para estudo do leitor. ●

Links

Rad Studio 2010 - Trial
http://downloads.embarcadero.com/free/delphi

IBExpert Personal Edition – Free
http://ibexpert.net/ibe/index.php?n=Main.DownloadFree

Firebird 1.5
http://www.firebirdsql.org/index.php?op=files&id=engine_156

Site do autor
http://www.devmedia.com.br/fabricio.kawata

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Delphi Prism e ASP.NET

Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 3

Guinther Pauli
guinther@devmedia.com.br
guinther.pauli@gmail.com
http://twitter.com/guintherpauli
http://gpauli.com
Desenvolve em Delphi há 15 anos, tendo utilizado a ferramenta desde a versão 1, passando pelo Turbo Pascal e Borland Pascal for Windows. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Delphi para Linux, palestrante em 5 edições da Borland Conference. Certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (C#, ASP.NET), utilizando a plataforma .NET desde a versão beta. Editor das Revistas ClubeDelphi, .NET Magazine Brasil e EASY .NET Magazine, professor de Pós-Graduação da UNIVEM-SP, na área .NET. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, Visual Studio, atuando também em projetos de migração, orientando empresas e desenvolvedores na adoção de melhores práticas, ministra treinamentos em .NET focados em equipes Delphi. Foi arquiteto e desenvolvedor ASP.NET do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

Do que trata o artigo
Lançado pela Embarcadero, o Delphi Prism é a nova forma de desenvolver aplicações para o .NET Framework e Mono utilizando o Object Pascal e o IDE do Visual Studio. Nesta série de artigos, veremos como utilizar o Delphi Prism para criar um site de E-Commerce usando a tecnologia ASP.NET.

Para que serve
O objetivo principal desta série de artigos é familiarizar os desenvolvedores Delphi Win32 com o ambiente de desenvolvimento do Delphi Prism, o IDE do Visual Studio, bem como introduzir o desenvolvimento de aplicações Web. Dessa forma, faremos vários comparativos entre os dois IDEs e também entre os Frameworks ASP.NET e VCL, servindo como um excelente guia para diminuir a curva de aprendizado ao se aprender ASP.NET baseando-se nos seus prévios conhecimentos em Delphi Win32 (Desktop).

Em que situação o tema é útil
Esta série será muito útil para quem nunca desenvolveu aplicativo algum para Web, nem mesmo teve um primeiro contato com o Delphi Prism / Visual Studio. Ao desenvolver aplicações para Web, ganhamos inúmeras vantagens se compararmos ao ambiente Desktop: Deploy simplificado, uso de computação nas nuvens, não há necessidade de instalações visto que os usuários acessam um mesmo servidor na internet, não há dependência do sistema operacional, pois usamos um browser, não há preocupação com backups nos clientes, nem altos custos para manutenções e visitas aos clientes etc.

Resumo do DevMan

No primeiro artigo desta série, tivemos uma introdução ao Delphi Prism, seus objetivos e propósitos. A seguir partimos para a criação do banco de dados no SQL Server que será usado ao longo desta série. Na segunda parte iniciamos a criação do Web Site, aproveitando para falar dos fundamentos do ASP.NET e Visual Studio, sempre comparando-os com a VCL e RAD Studio, com o intuito de facilitar o aprendizado. Vimos a estrutura básica de um Web Site e iniciamos a construção do layout básico do site de E-Commerce (chamado "Computer Store"), bem como o sistema de navegação. Neste artigo vamos construir a página principal do site, exibindo a lista de produtos, mostrando como fazer a conexão com o banco de dados.

Existem várias formas de acessar dados no ASP.NET, o que é motivo para muita confusão principalmente para quem está começando. Só para citar por cima, hoje é possível usar ADO.NET puro, DataSets Tipados com TableAdapters, criar uma camada customizada de acesso (uma DAL – Data Access Layer), usar o Framework de persistência e mapeamento objeto relacional da Microsoft, o ADO.NET Entity Framework, ou ainda o NHibernate, entre outros. Pessoalmente prefiro, em meus projetos e treinamentos, utilizar uma abordagem interessante, robusta e profissional: separar a aplicação em camadas, de forma que as classes de negócio, apresentação (forms) e domínios (as classes que mapeiam as tabelas do banco) não dependam de uma tecnologia de persistência em específico, usando aí também um pouco de Interfaces, Generics e classes POCO (Plain Old CLR Objects). Dessa forma, é possível desenhar uma solução e aplicar um modelo de persistência de acordo com a necessidade, ou mesmo mudar de tecnologia de acesso sem sofrer um impacto drástico.

No entanto, é fato que os desenvolvedores Delphi Win32 não estão acostumados a utilizar Frameworks de persistência. Ou alguém aí cria uma classe chamada TProduto para mapear uma tabela Produto do banco de dados em uma aplicação Delphi Win32? Eu não (neste tipo de aplicação). Coloco um ClientDataSet e pronto, uma classe só que manipula qualquer tabela de qualquer banco ou outras fontes. É 100% mais produtivo e o resultado é o mesmo. No entanto, quando pensamos em aplicações orientadas a objetos, principalmente em soluções construídas sobre plataformas como Java e .NET, devemos levar em consideração o uso de programação orientada a objetos com persistência e mapeamento objeto / relacional.

Neste curso, no entanto, como o objetivo principal é reaproveitar os conhecimentos adquiridos ao longo do desenvolvimento com Delphi Win32 /VCL ao desenvolver para Web com ASP.NET, vamos utilizar um recurso introduzido no ASP.NET 2.0 / Visual Studio 2005, os DataSources (que não tem nada a ver com o TDataSource da VCL). Neste caso, vamos usar um SqlDataSource para acessar os dados do banco de dados. O SqlDataSource é um controle não visual que é uma "casca extremamente fina" sob o ADO.NET (a base de todo tipo de acesso a dados no .NET, e também a forma mais rápida de acesso). O SqlDataSource em si não faz muita coisa, tudo o que ele precisa fazer, como conectar bancos de dados, executar comandos SQL, recorre para classes do ADO.NET como SqlConnection e SqlCommand (no caso de ser utilizado o provider para acesso ao SQL Server).

O SqlDataSource é um "faz tudo", ele encapsula em uma única estrutura tudo o que você precisa para manipular dados de uma tabela. Ele conecta, recupera dados, atualiza dados, faz cache (caso seja especificado) tudo de forma automática (como faz o DataSetProvider com ClientDataSet no dbExpress). Além disso, controles de tela como GridView e DataList podem se ligar diretamente a DataSources, através da propriedade DataSourceID, minimizando ainda mais a codificação. Por exemplo, ao editar dados em tela e solicitar uma atualização, o próprio SqlDataSource gera os comandos e parâmetros para execução de comandos SQL no banco. No ASP.NET 1.x devíamos fazer isso "no braço". Dessa forma, o uso do controle é muito semelhante ao que já estamos acostumados a fazer no Delphi Win32, considerando que você utilize dbExpress, DataSetProvider, ClientDataSet e controles Data-Aware.

Também veremos como configurar o acesso a dados no IDE do Delphi Prism (Visual Studio), de forma a manipular as informações do BD a partir do próprio IDE, como faz o Data Explorer do Delphi Win32. Falarei um pouco sobre providers para conexão e darei dicas inicias de performance. Além disso, nesta etapa do curso vamos aprender a exibir dados em controles "Data-Aware". Aqui no ASP.NET qualquer coisa é "Data-Aware", como veremos no exemplo, pois usamos o conceito de DataBinding. Nós vamos utilizar o controle DataList para exibir na tela principal os produtos cadastrados no banco de dados. Ele utilizará um SqlDataSource para obter as informações necessárias do SQL Server.

Configurando a conexão com o banco de dados no IDE

Com o IDE do Delphi Prism aberto, clique em View > Server Explorer. Esta janela tem papel equivalente ao Data Explorer do IDE do Delphi Win32, com a diferença que ela não serve apenas para manipular dados de banco de dados, é possível através dela acessar importantes itens de um servidor local ou remoto, como ferramentas administrativas (log de eventos, serviços, contadores de performance) etc.

Vamos adicionar uma conexão ao banco de dados criado na primeira parte deste curso. Dê um clique de direita sobre o item Data Connections e escolha a opção Add Connection (**Figura 1**). Será aberta a janela mostrada na **Figura 2**, onde vamos informar os dados relativos à conexão. Em Data Source vamos manter a opção Microsoft SQL Server (SqlClient), esse é o provider nativo para acesso ao SQL Server. Jamais use ODBC ou OLEDB para acesso a esse banco, pois adicionam uma "camada" extra à aplicação e significam perda de performance.

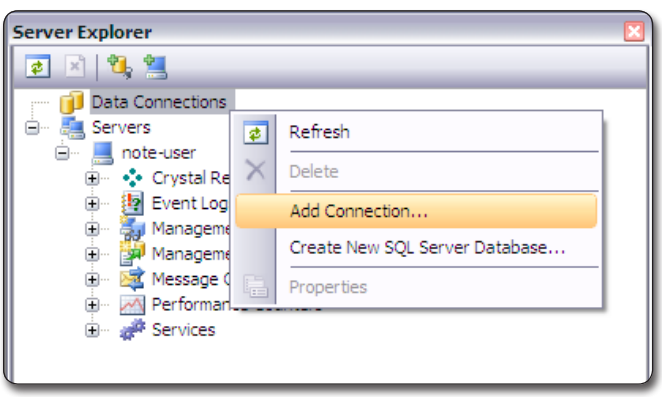


Figura 1. Adicionando uma conexão

Na opção Server Name informe o nome da instância do servidor SQL Server que estamos trabalhando, neste caso .\SQLEXPRESS. Em Select or enter a database name escolha o nosso banco LOJA, que está devidamente registrado. Note que é possível conectar diretamente a um arquivo de dados do MSSQL, através da opção Attach a database file. Para testar se tudo está OK clique no botão Test Connection.

Nota do DevMan

Clicando no botão Change é possível escolher outro provider para acesso a outro tipo de banco de dados, como mostra a **Figura 3**. Nela você pode notar uma entrada interessante: DataSnap. É possível criar uma página Web em ASP.NET que acesse uma camada intermediária de lógica de negócio (Servidor de Aplicação). Essa é uma saída bastante interessante para você ter um aplicativo que tem uma interface Desktop Win32 criada em Delphi nativo compartilhando regras e SQLs com uma interface ASP.NET. Porém, aqui vai uma dica que a experiência confirma: em aplicações ASP.NET caso não seja extremamente necessário não adicione uma camada extra física à solução, isso acarreta perda de performance, pois processos precisarão se comunicar para trocarem dados (o que chamamos de marshalling, na comunicação entre processos - "out of proc"). Na Web a regra é simples: banco de dados, servidor Web e browser, mais nada. Se for necessário desenvolver uma aplicação em camadas, uma boa prática de desenvolvimento, faça uma separação lógica, criando, por exemplo, uma classe de acesso mas que resida no mesmo processo da aplicação Web. Para mais informações veja sessão Links.

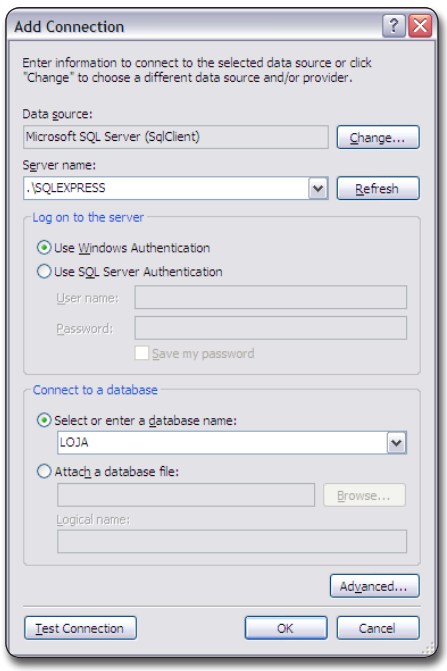


Figura 2. Parâmetros de conexão

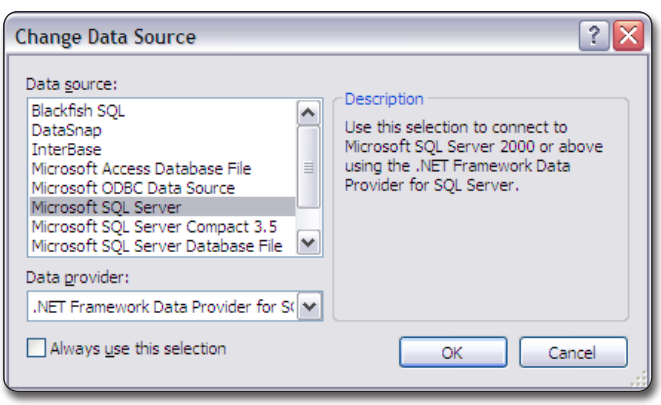


Figura 3. Provider de conexão

Nota do DevMan

É possível criar páginas, classes e outros itens em Delphi Prism e mesclar com itens criados em outras linguagens, como C#, em um mesmo Web Site ou projeto (mesmo em um nível binário). Isso é possível graças a todas as linguagens obedecerem uma mesma especificação, chamada CLS (Common Language Specification).

A partir deste ponto a conexão estará registrada no IDE do Visual Studio e estará disponível para qualquer aplicação que você venha a criar. Faça alguns testes, expanda o nó principal do BD e veja que temos acesso a toda estrutura de objetos do banco de dados, como Tabelas, Views, Stored Procedures, Diagramas etc. Você pode utilizar o IDE do Visual Studio para gerenciar praticamente qualquer parte do BD, visualizar dados, executar query's, criar novas tabelas, SPs, Views, quase tudo o que o SQL Server Management Studio Express faz. Veja por exemplo, na **Figura 4**, que podemos visualizar o diagrama ER do banco de dados criado na primeira parte do curso.

Criando a página principal herdando da Master Page

Vamos voltar ao nosso Web Site. Clique na opção File > Open Web Site e selecione o diretório do nosso projeto (c:\LOJA). No Solution Explorer clique de direita no projeto e escolha Add New Item. Na janela que aparece (**Figura 5**) escolha o template Web Form, dando a este formulário o nome de Home.aspx. Veja que escolhemos a linguagem Oxygene (Delphi Prism) como code-behind para a página, mas graças a CLS poderíamos codificar parte do site em uma linguagem e parte em outra, por exemplo.

Um ponto importante ainda na **Figura 5** é a marcação da opção Select master page, em destaque. Ela indica que vamos herdar a parte visual definida na nossa Master Page, criada na parte anterior deste curso. Na tela seguinte (**Figura 6**), o VS vai apresentar a lista de todas as Master Pages disponíveis no projeto, apenas confirme selecionando a única opção mostrada (arquivo Masterpage.master).

Neste momento será exibida a página no designer. Na parte inferior você pode escolher se deseja visualizar o Web Form na forma de Design ou código ASPX (ou Split, que mostra ambos simultaneamente). Vá ao modo Source e repare a diretiva Page da nossa página Home, onde podemos ver o vínculo criado com a Master Page, através do atributo MasterPageFile. Indo para o code-behind da página Home, você pode comprovar que a "herança" não é pura da orientação a objetos, como comentei na parte anterior deste artigo. Veja que o Web Form herda da classe base Page, e não da Master. Ou seja, a herança é só visual. Você pode verificar isso no código da **Listagem 1**.

Configurando o acesso a dados com SqlDataSource

Vamos agora mostrar a lista de produtos cadastrados na página Home. Por enquanto, não vamos permitir nenhuma pesquisa nem filtro, vamos mostrar todo o conteúdo da tabela de produtos. Dentro do ContentPlaceHolder coloque um controle

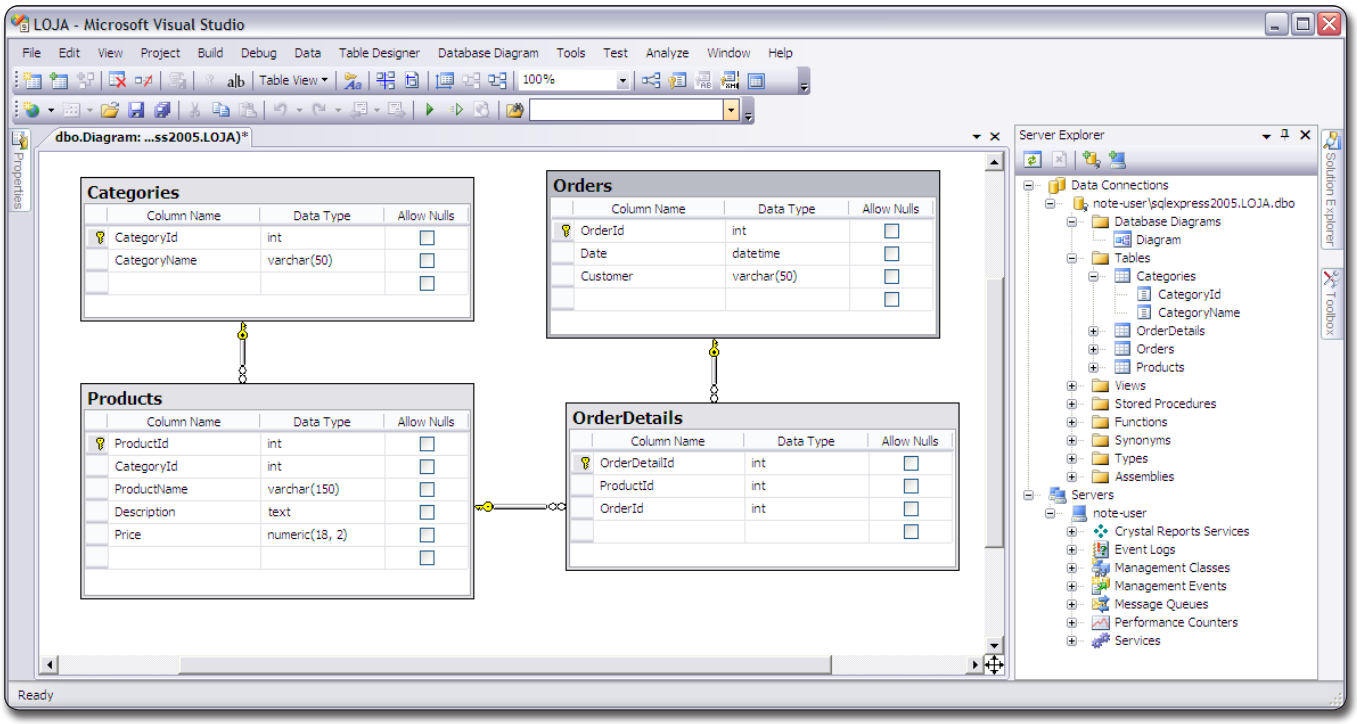


Figura 4. É possível gerenciar praticamente todo o BD a partir do próprio IDE do VS

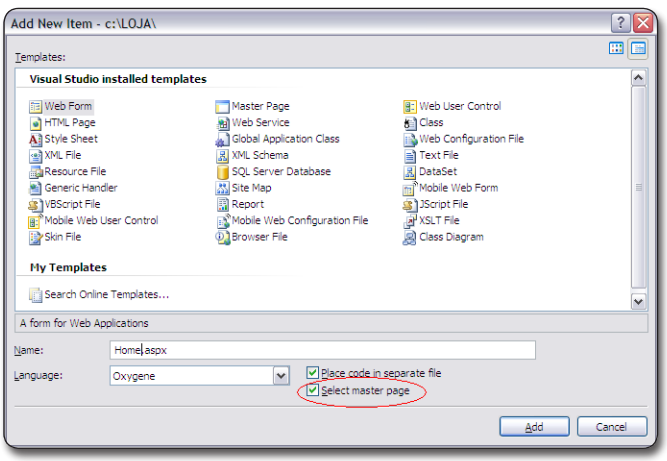


Figura 5. Adicionando um Web Form herdando da Master Page

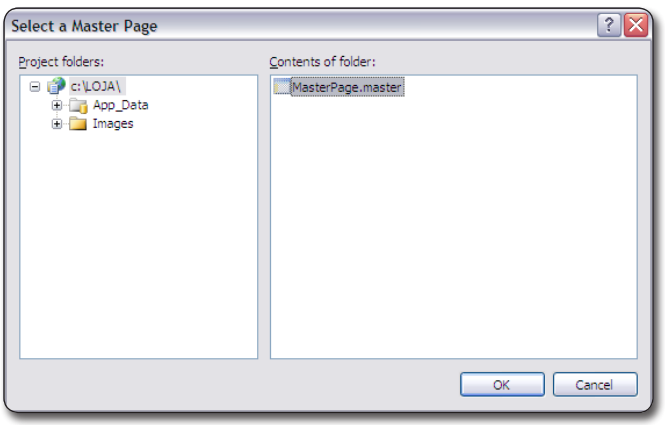


Figura 6. Escolhendo a Master Page

Listagem 1. Diretiva Page apontando para a Master Page, e code-behind do Web Form

```
[ASPX]
<%@ Page Language="Oxygene"
MasterPageFile="~/MasterPage.master"
AutoEventWireup="true"
CodeFile="Home.aspx.pas"
Inherits="Home"
Title="Home"%>

[Code-behind .PAS]
// Repare que Home não herda da Master
Home = public partial class(System.Web.UI.Page)
```

DataList da guia Data (**Figura 7**). Na SmartTag do controle que deve aparecer, escolha a opção Choose Data Source > New Data Source para abrir o assistente de conexão.

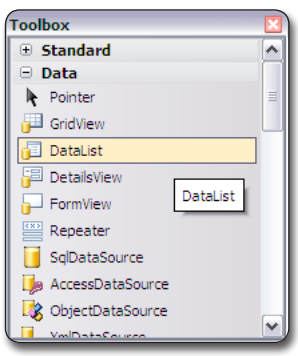


Figura 7. DataList

Na tela que aparecerá (**Figura 8**), escolha a opção Database, o que irá criar um SqlDataSource. Na tela seguinte escolha a

conexão previamente criada no IDE do VS (**Figura 9**). Note que neste ponto, caso não tivéssemos configurado uma conexão, poderíamos fazê-la aqui através da opção New Connection. Abaixo será mostrada a string de conexão (Connection String) com o BD. Na tela seguinte apenas confirme que deseja salvar estas configurações de acesso no Web.Config, com o nome padrão de LOJAConnectionString.

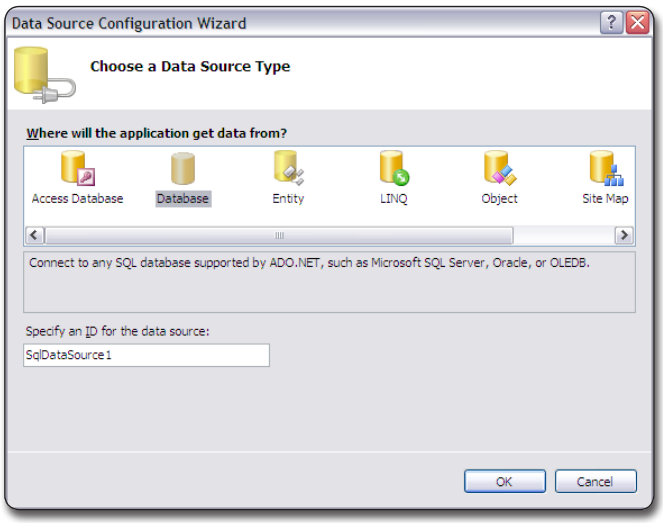


Figura 8. Opção Database acessa um banco relacional e irá criar um SqlDataSource

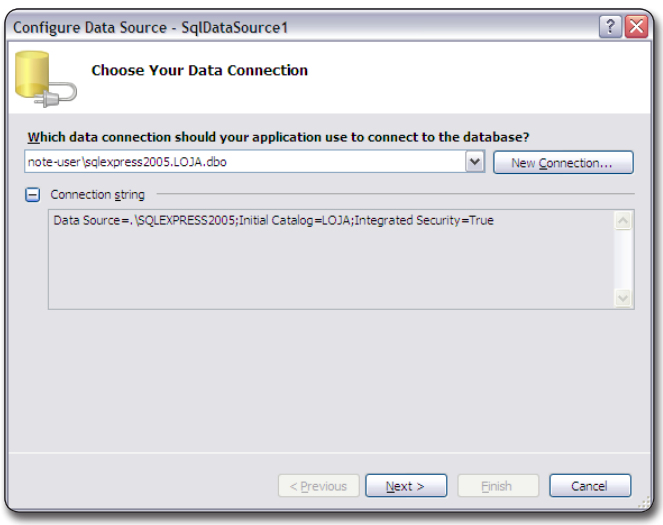


Figura 9. Escolhendo a conexão e visualizando a Connection String

Na tela seguinte (**Figura 10**) podemos configurar a instrução SQL que queremos ter para nosso SELECT, ou pedir para o VS gerar automaticamente para nós, apenas selecionando os campos como mostrado. Finalize o assistente confirmando as demais opções.

Configurando o DataList

Repare que o DataList já apresenta agora algumas informações sobre o DataSource que estamos trabalhando, como por exemplo o nome das colunas. No entanto, os dados reais só serão exibidos quando a aplicação for executada. O editor HTML designer do VS não suporta o recurso de "live-data" com o Delphi Win32. No entanto você pode visualizar os dados retornados em design time no próprio assistente do passo anterior.

Nota

Nesta janela é possível usar um Query Builder para visualmente definir as instruções SQL, por exemplo, arrastando campos de uma tabela para outra para criar um join. Além disso, podemos usar a opção Advanced para pedir ao VS para gerar automaticamente as instruções SQL de atualização, como Update, Insert e Delete, além de configurar opções de resolução de conflito em updates (concorrência otimista e pessimista). Não vamos precisar essas opções simplesmente porque em nossa tela principal vamos apenas exibir dados dos produtos (pesquisas), sem necessidade de alterações.

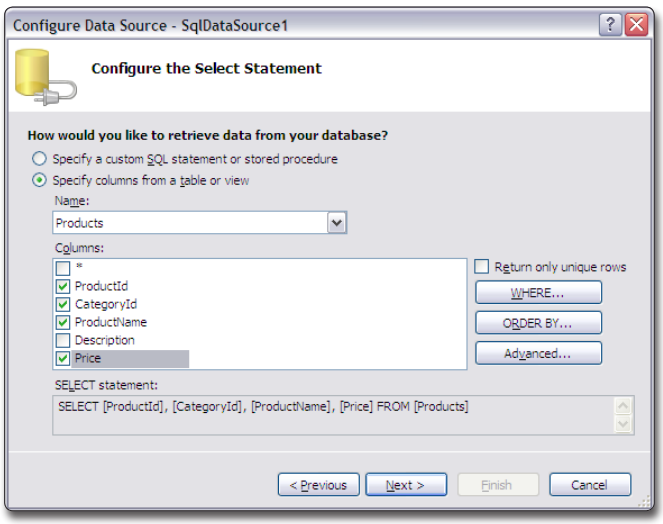


Figura 10. Definindo visualmente a instrução Select

Vamos configurar o layout do DataList, definindo primeiramente seu RepeatDirection para Horizontal e RepeatColumns para 3, o que vai exibir três produtos por linha. A seguir, na SmartTag do controle (a seta que aparece no canto superior direito) escolha a opção Edit Templates para acessar os controles internos já definidos, como os Labels. Pense em "editar templates" em um processo semelhante ao de edição de colunas individuais de um controle DBGrid da VCL do Delphi Win32. Porém aqui isso se aplica a praticamente qualquer controle da guia Data.

Remova os campos ProductID, ProductName e CategoryID, pois usaremos outros controles que não o Label para exibir as informações dessas colunas. Por exemplo, vamos exibir o nome do produto como um Link. Ao invés do ID do produto, vamos mostrar sua imagem. Comece adicionando ao template um HyperLink (guia Standard). Na opção Edit Databindings diga que o texto do Hyperlink será o nome do produto (**Figura 11**). Repare aqui o conceito de DataBind, como explicou o DevMan, poderíamos vincular qualquer propriedade do controle a um valor vindo do banco. O que fizemos foi algo semelhante ao configurar a propriedade DataField de um controle Data-Aware do Delphi Win32.

Nota do DevMan

DataBinding

Faça um teste. Selecione o ProductIdLabel e através da sua SmartTag repare que este Label está fazendo DataBind (ligação) para o campo ProductName do DataSource. Isso é semelhante ao conceito de controle "Data-Aware" do Delphi Win32, porém aqui não precisamos de controles diferentes para acessar dados (por exemplo, uma versão do Edit "normal" e o DBEdit para uso com BD). Aqui qualquer coisa pode exibir dados de qualquer outra fonte, não necessitando que essas informações venham necessariamente de um BD. Por exemplo, podemos vincular o Text de um TextBox ao Text de um Label (ou Hint - Tooltip, ou Cor, ou estilo, qualquer coisa). Se você alterar o texto da caixa, o Label exibirá o texto sem que para isso se faça uma atribuição em código Delphi. Indo ao direto ao ponto, ao invés de fazer isso em algum evento:

Label1.Text := TextBox1.Text;

Podemos fazer no ASPX:

<asp:Label ID="Label1" runat="server" Text='<%# TextBox1.Text %>'

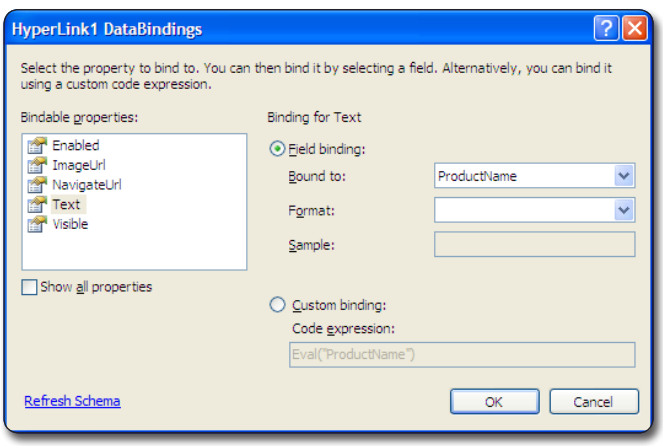


Figura 11. Texto do HyperLink exibirá o nome do produto

Para o campo NavigateUrl precisamos fazer um binding customizado. Marque a opção Custom binding e em Code Expression (essas opções podem ser vistas na mesma janela da **Figura 11**) digite:

```
Eval("ProductId","~/ShowProduct.aspx?ProductId={0}")
```

Isso faz com que, quando o HyperLink (que agora mostrará o nome do produto) for clicado, o usuário seja redirecionado para a página que mostra os detalhes do produto, passando o ID do produto selecionado diretamente na Query String (URL). Neste outro Form, que construiremos mais adiante no curso, vamos mostrar todos os detalhes do produto e permitir que o usuário adicione-o ao carrinho de compras.

Agora adicione um controle Image no template. Acione seus DataBindings, selecione ImageUrl e na propriedade de customização do binding informe:

```
Eval("ProductId","~/Products/{0}.jpg")
```

Isso faz com que a imagem aponte para uma URL, no caso uma imagem que estará no disco, por exemplo, 1.jpg, 2.jpg etc. Repare que para isso devemos colocar as respectivas imagens dentro da pasta Products para representar os produtos do banco de dados, usando a nomenclatura indicada ([ProductID].jpg). O parâmetro {0} vai fazer com que o ProductID seja colocado na URL no momento do DataBind.

Nota

É possível armazenar imagens no BD e mostrá-las em controles do ASP.NET, como DataList e GridView, porém essa não é a abordagem que utilizaremos aqui, por questões de desempenho. Recuperar todas as imagens de produtos a cada request é uma operação muito lenta, é melhor fazer cache em disco ou memória, gerar thumbnails, e exibir diretamente em imagens na tela.

Seu DataList deve estar semelhante ao mostrado na **Figura 12**. Para finalizar os trabalhos na edição dos templates clique na SmartTag do controle e escolha End Template Editing. A **Listagem 2** mostra o código gerado para o arquivo ASPX da página, finalizada para esta etapa. A **Figura 13** mostra o site rodando (Ctrl+F5).

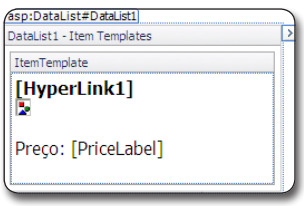


Figura 12. DataList em modo de edição de templates

Listagem 2. Código ASPX da página principal (Home.aspx)

```
<%@ Page Language="Oxygene" MasterPageFile="~/MasterPage.master"
  AutoEventWireup="true" CodeFile="Home.aspx.pas" Inherits="Home"
  Title="Home" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
  Runat="Server">
   <br />
   <asp:DataList ID="DataList1" runat="server" DataKeyField="ProductId"
     DataSourceID="SqlDataSource1" RepeatDirection="Horizontal"
     RepeatColumns="3">
     <ItemTemplate>
       <asp:HyperLink ID="HyperLink1" runat="server" Font-Bold="True"
         style="font-family: Tahoma" Text='<%# Eval("ProductName") %>'>
         </asp:HyperLink>
       <br />
       <asp:Image ID="Image3" runat="server"
         ImageUrl='<%# Eval("ProductId","~/Products/{0}.jpg") %>' />
       <br style="font-family: Tahoma" />
       <br style="font-family: Tahoma" />
       <span style="font-family: Tahoma">Preço: </span>
       <asp:Label ID="PriceLabel" runat="server" Text='<%# Eval("Price") %>'
         style="font-family: Tahoma" />
       <br />
       <br />
     </ItemTemplate>
   </asp:DataList>
   <asp:SqlDataSource ID="SqlDataSource1" runat="server"
     ConnectionString="<%$ ConnectionStrings:LOJAConnectionString %>"
     SelectCommand="SELECT [ProductId], [CategoryId], [ProductName],
     [Price] FROM [Products]">
   </asp:SqlDataSource>
</asp:Content>
```

Nota

Ctrl+F5 (Start Without Debugging) roda a aplicação sem levantar o Debugger, e equivale ao Run Without Debugging presente desde a versão 8 do IDE do Delphi (e escondido nas versões anteriores, como a 6 e 7). Para rodar com o Debugger, pressione F5 e confirme na caixa que solicitará uma alteração no arquivo de configuração. O F5 do VS equivale ao F9 do Delphi Win32.

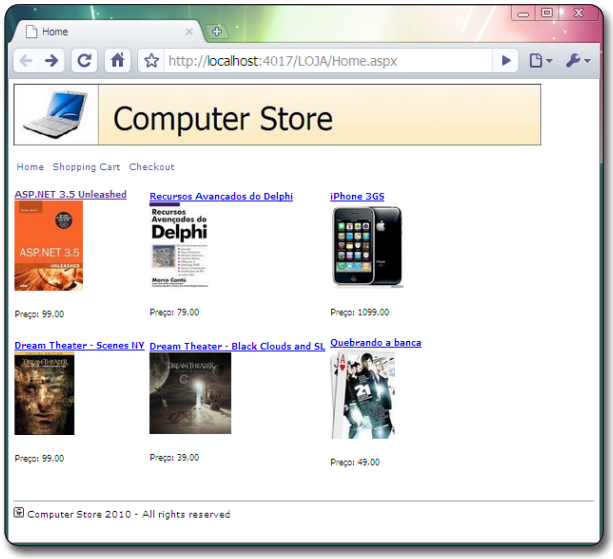


Figura 13. Página principal em execução

Conclusão

Reparou algo até aqui? Nós não digitamos nenhuma linha de código Delphi! E já temos uma tela principal totalmente funcional. Isso prova que, programar com ASP.NET e seus Server Controls, é tão produtivo quanto usar a VCL do Delphi com todos seus componentes prontos. Note a clara relação e similaridade com o desenvolvimento Win32, o conceito de arrastar e soltar (drag & drop), o uso de controles, configuração de propriedades, uso de assistentes, geração automática de código etc. Você terá mais certeza disso nas próximas etapas, quando vamos iniciar a escrita de código em Prism e adicionar mais funcionalidades interessantes ao nosso Computer Store, como um carrinho de compras. Grande abraço, sucesso com o ASP.NET e até a próxima edição. ●

Links

Site oficial do Delphi Prism
http://www.embarcadero.com/products/delphi-prism

Wiki do Delphi Prism
http://prismwiki.codegear.com/en/Main_Page

Download do Delphi Prism
https://downloads.embarcadero.com/free/delphi_prism

Site oficial do ASP.NET
http://www.asp.net

Visual Studio
www.microsoft.com/visualstudio

Blogs do autor
http://twitter.com/guintherpauli
http://gpauli.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

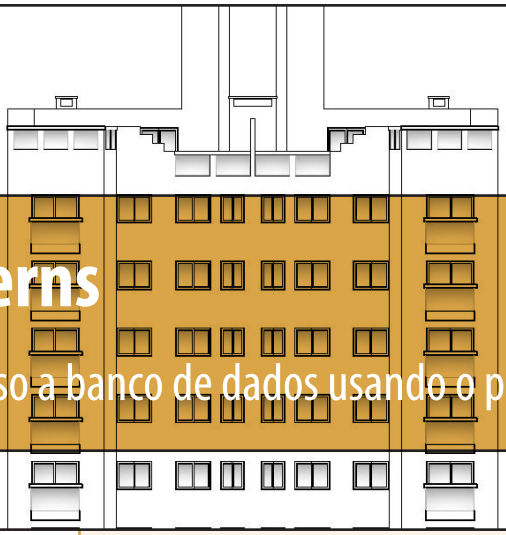
www.devmedia.com.br/clubedelphi/feedback

Faça um *up grade* em sua carreira.

Em um mercado cada vez mais focado em qualidade ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lança para os desenvolvedores brasileiros sua primeira revista digital totalmente especializada em Engenharia de Software. Todos os meses você irá encontrar artigos sobre Metodologias Ageis; Metodologias tradicionais (*document driven*); ALM (*application lifecycle management*); SOA (aplicações orientadas a serviços); Analise de sistemas; modelagem; Métricas; orientação à objetos; UML; testes e muito mais. **Assine já!**

DevMedia group

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP



Design Patterns

Centralizando o acesso a banco de dados usando o padrão Singleton

Do que trata o artigo

Os padrões de projeto foram criados na década de 70 pelo arquiteto Christopher Alexander, para solucionar problemas comuns da construção civil. Mais tarde estes padrões de projeto ou Design Patterns, como são chamados, passaram a ter o objetivo de estabelecer como o próprio nome diz, padrões e normas, mas, agora com a função de serem aplicados na criação de projetos de software. Neste artigo, iremos conhecer os 5 (cinco) padrões de projetos da fase de criação de um software, que são: Abstract Factory, Builder, Factory Method, Prototype e Singleton. E após termos conhecido superficialmente o que propõe cada um deles teremos uma explanação do método Singleton aplicado com PHP.

Para que serve

Aplicar boas práticas de desenvolvimento nos projetos, conhecer os Design Patterns e possibilitar que o leitor não só saiba definir e conhecer os padrões de projeto, mas saiba como aplicá-los na prática ao longo dos seus projetos em PHP.

Em que situação o tema é útil

Este artigo é útil para desenvolvedores que possuem certo conhecimento na linguagem PHP Orientado a Objetos, porém, desenvolvem suas aplicações sem seguir nenhum padrão de desenvolvimento, sem se preocupar com a necessidade de manutenção futura desta aplicação, ou até mesmo com a necessidade de outras pessoas trabalharem em conjunto no mesmo projeto. É de grande valia para quem tem interesse em otimizar seus aplicativos para ter maior controle do desenvolvimento e maior usabilidade dos códigos escritos em suas aplicações.

Dheyson Wildny
dheysonws@gmail.com
Blog: http://www.devfranca.com.br
Tem 5 anos de experiência na área de desenvolvimento e arquitetura de software. Graduado em Sistemas de Informação pela Universidade de Franca – Unifran, trabalho de conclusão de curso destacado dentre os melhores da universidade no ano de 2007. Analista de Sistemas da empresa Irroba Multimedia. Especialista em desenvolvimento PHP e ASP, atuando principalmente na criação e desenvolvimento de projetos voltados a lojas virtuais e gerenciamento de produtos. Co-criador da comunidade Dev Franca, que auxilia jovens desenvolvedores a encontrar o melhor caminho e as melhores práticas no desenvolvimento de aplicativos tanto Web quanto Desktop.

Resumo do DevMan

Este artigo irá resumir de forma clara o que são estes cinco padrões de criação, mostrando aos desenvolvedores que é muito importante manter uma rotina e um padrão, para agregar maior valor em suas aplicações e para que o desenvolvedor esteja sempre preparado para desenvolver aplicações robustas e de fácil manutenção. Ao final deste artigo além de saber o que cada padrão desses vem propor, o desenvolvedor estará apto a centralizar todo acesso ao banco de dados de sua aplicação em uma única classe, que fará todo trabalho de conexão com o sistema gerenciador de banco de dados (SGBD). O principal foco deste artigo é mostrar como criar corretamente uma classe de conexão para sua aplicação respeitando os Creational Patterns (padrões de criação), sempre que for necessário fazer umas das quatro operações básicas com o banco de dados, as chamadas CRUD (Create, Retrieve, Update and Delete).

Ao projetar um sistema, o analista deve se preocupar em desenhar o projeto de uma maneira em que este mesmo projeto possa ser desenvolvido em várias linguagens de programação, por exemplo, C#, SmallTalk, Java, Asp.Net, PHP etc. e ser compreendido pelos desenvolvedores, além de apontar o caminho correto a seguir durante o desenvolvimento do mesmo. Daí vem aquela velha pergunta: mas que benefícios eu terei em aplicar os tais Patterns em meu projeto?

• Bom, aqui vão alguns destes benefícios:

• Facilidade de comunicação entre a equipe de desenvolvimento: quando todos na equipe seguem a mesma linha de raciocínio, fica muito mais fácil desenvolver o projeto, pois quando é necessário um programador dar continuidade no trabalho de outro, o mesmo não fica perdido nas linhas de programação escritas;

• Maior manutenibilidade do sistema: quando se torna necessário aplicar qualquer tipo de manutenção no sistema o problema é minimizado, pois todo software foi desenvolvido seguindo um mesmo padrão;

• Legibilidade do código: o tempo para ler e compreender todo código se torna muito menor, pois o código segue uma organização pré-definida, ou seja, segue um padrão.

Assim que alguns programadores e analistas vêem estes benefícios, uma das reações que têm é a que irá adotar todos os Patterns em todas suas aplicações. E não é bem este o caminho, os padrões são criados com o intuito de corrigir algum tipo de problema recorrente, e isso não só na informática, mas em todos setores de nossa sociedade. A primeira coisa a se fazer é descobrir se sua aplicação possui algum tipo de problema que deva ser corrigido, após identificar o problema é hora de tentar estabelecer algum padrão para acabar com este problema encontrado.

Como foi dito anteriormente, neste artigo o principal intuito é fazer com que o desenvolvedor entenda alguns dos principais padrões de criação, para que assim possa analisar suas aplicações e identificar se há necessidade de estabelecer algum ou alguns deles em seu desenvolvimento. Vamos dar início aos conceitos, definições e compreendê-los.

Abstract Factory

Esse é um padrão que na maioria das vezes é compreendido com maior facilidade pela comunidade, pois é muito semelhante a um sistema que já é bastante utilizado que é a criação de interfaces. Este padrão é utilizado quando temos duas ou mais classes semelhantes, mas que possuem características específicas (chamadas classes concretas), daí então criamos uma classe que define um "molde" de como as classes finais devem ficar (a classe base).

Imagine a seguinte situação, precisamos definir duas classes para desenhar algumas figuras geométricas, por exemplo, um círculo e um quadrado e em ambas as classes deveremos implementar um método para desenhar e outro para centralizar o objeto na tela. Teremos de concordar que apesar das duas serem formas geométricas e as duas possuírem um método chamado desenharForma e outro setCentro, não quer dizer que estes métodos serão desenvolvidos da mesma forma. É quando entra o Abstract Factory. Para que estas duas classes semelhantes sejam desenhadas uniformemente, é criada uma classe abstrata para definir como estas classes deverão ser implementadas, ou para um melhor entendimento, é criada uma classe modelo. Nesta classe ficam estabelecidos os tipos e os nomes dos atributos e métodos a serem utilizados pelas classes finais. Vejamos o exemplo demonstrado na **Listagem 1**.

Nota do DevMan

Os padrões de projeto ganharam mais força com o livro "Design Patterns: Elements of Reusable Object-Oriented Software", originalmente escrito pela famosa e referenciada GoF (Gang of Four). Além dos padrões de criação temos ainda os padrões estruturais e comportamentais. A divisão destes padrões é relacionada da seguinte forma:

Criacionais:
Abstract Factory, Builder, Factory Method, Prototype, Singleton;

Estruturais:
Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy;

Comportamentais:
Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method e Visitor.

Listagem 1. Demonstração de Abstract Factory

```
01 <?php
02
03 abstract class FormaGeometrica {
04       protected $x
05       protected $y
06
07   public function setCentro($x, $y, $width, $height) {
08
09       $this->x = (($x/2)-($width/2));
10       $this->x = (($x/2)-($width/2));
11
12      }
13
14      abstract function desenharForma();
15 }
16
17
18
19 class Quadrado extends FormaGeometrica {
20       function desenharForma() {
21       //Aqui é definido o comportamento deste método nesta classe
22            ...
23      }
24 }
25
26
27 class Circulo extends FormaGeometrica {
28       function desenharForma() {
29       //Aqui é definido o comportamento deste método nesta classe
30            ...
31      }
32 }
34 ?>
```

Na Linha 3 podemos ver que foi definida nossa classe abstrata chamada FormaGeometrica. Nesta classe são definidos

dois atributos com escopo protegidos (protected), ou seja, são atributos que só podem ser acessados pela própria classe criadora ou por classes derivadas desta, possui também um método chamado setCentro, que posiciona a imagem ao centro da tela. Este método para centralizar, não foi definido como abstrato porque tanto o quadrado quanto o círculo poderão se utilizar desta mesma rotina para se centralizar, portanto não há necessidade de reescrevê-lo nas classes derivadas, e por último declaramos somente o escopo do método chamado desenharForma. Este escopo fica definido como abstrato para garantir que o mesmo seja implementado em cada classe derivada, cada um com sua rotina específica, mas ambos com mesmo nome para garantir a padronização no software.

Já nas Linhas 19 e 27 são definidas as classes Quadrado e Circulo respectivamente, ambas herdando métodos da classe abstrata FormaGeometrica, e nelas o método desenharForma deverá ser especificado, cada um a sua maneira. E nas duas classes tanto os atributos $x e $y quanto o método setCentro poderão ser acessados normalmente.

Finalizando o Pattern, criamos então uma classe que fabrica as classes concretas internamente, mas que devolve tipos da classe base. Quem precisar instanciar essas classes concretas não deve fazer isto diretamente através de seus construtores, mas sim solicitando a essa fábrica, daí o nome "Factory". Normalmente também podemos fazer uma classe abstrata para representar uma fábrica base, e classes fábricas concretas para representar e devolver cada uma das classes concretas alvo. No exemplo da **Listagem 1**, teríamos então uma classe base FormaGeometricaFactory e duas descendentes, Quadrado Factory e CirculoFactory.

Builder

Chegamos ao segundo padrão, o chamado Builder (construtor). Ao passo que já entendemos bem o que é o Abstract Factory, não teremos muitas dificuldades em entender o que é, e como funciona este padrão. Seguindo até certo ponto o mesmo raciocínio do padrão anterior, podemos dizer que o Builder é um padrão que especifica como um grupo de classes que tratam um mesmo objeto complexo devem ser montadas. Para definir qual é a função de cada classe deste grupo podemos dividi-lo em 4 elementos básicos, que são:

- Director: cria um objeto utilizando da interface do Builder;
- Builder: especifica a interface para o construtor de partes do objeto;
- Concrete Builder: define a implementação da interface criada pelo Builder e mantém a implementação da interface que será utilizada pelo Product;
- Product: Cria e organiza as classes constituintes do objeto em questão;

Vamos entender o que estes quatro elementos querem nos trazer de bom, e para entender melhor o que é cada um desses nada melhor que explicar com uma demonstração prática. Por exemplo, este padrão poderia ser utilizado em uma situação em que tenhamos que construir uma classe complexa que possa retornar vários objetos distintos dentro de uma mesma família, por exemplo, temos que construir uma classe para nos retornar objetos dos tipos, casa, apartamento e etc. Então criamos uma classe Director que irá direcionar como esses objetos deverão ser criados, outra classe que será a Builder, que é uma classe abstrata que apenas define quais métodos e atributos deverão ser implementados para construção desses objetos, como vimos no padrão anterior. Depois temos a classe ou as classes Concrete Builder, que especificam o comportamento dos métodos definidos na classe Builder e possuem um método que retorna o objeto propriamente dito, essa classe é responsável por definir o que é preciso para uma construção ser uma casa ou apartamento por exemplo. E finalmente temos as classes Product que organizam os métodos especificados e fazem o toque final para que então possamos consumir o objeto criado. Vamos analisar a **Figura 1** para entender melhor, com outro exemplo.

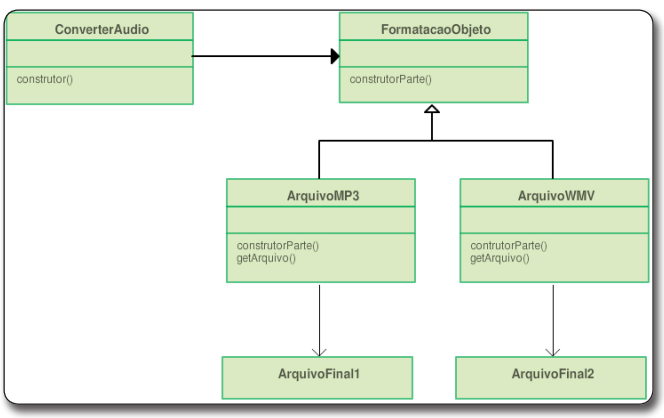


Figura 1. Demonstração de Builder

Na **Figura 1** criei o exemplo de um conjunto de classes para retornar um arquivo de áudio em vários formatos, neste caso .wmv e .mp3. Utilizando a mesma lógica do exemplo citado anteriormente a classe ConverterAudio é nossa Director, a classe FormatacaoObjeto é a Builder, ArquivoMP3 e ArquivoWMV são as Concrete Builders e as ArquivoFinal1 e ArquivoFinal2 são os objetos retornados ou Products. Em um exemplo real, poderíamos facilmente utilizar o Abstract Factory para resolver a mesma situação. Mas em softwares de larga escala e com grande número de desenvolvedores trabalhando no mesmo projeto este padrão pode ser bastante utilizado, pois cada desenvolvedor focaria seu trabalho na construção de um determinado objeto apenas.

Factory Method

Este padrão é utilizado em casos em que o projeto possui um grupo com várias classes semelhantes, mas com implementações distintas. Geralmente todas as classes deste grupo herdam algumas características de uma classe tida como "classe pai", e estas classes possuem métodos iguais com implementações diferentes, dando características únicas a cada objeto criado.

Uma forma simples de compreender como funciona este padrão é imaginar a seguinte situação: Seu projeto possui uma "classe pai" chamada Aeronave, e possui algumas "classes

filhas" chamadas de Cargueiro, MonoMotor e Jato. As três classes filhas são a abstração de alguns tipos de aeronaves encontrados no mundo real, pois bem, agora imagine que seja necessário saber o valor de cada um destes modelos. Sem utilizar um padrão faríamos da seguinte maneira, instanciaríamos um objeto para cada classe e só depois solicitaríamos o método que nos traga o preço.

Utilizando-se do padrão Factory Method teríamos uma outra classe que poderia se chamar AeronaveFactory, que seria a responsável por instanciar cada classe de acordo com nossa necessidade e nos retornaria um objeto diferente para cada tipo de solicitação. Ou seja, instanciaríamos um objeto da classe AeronaveFactory, especificaríamos o tipo de aeronave e solicitaríamos o método getPreco do objeto criado, este, buscaria o preço da classe necessário e nos retornaria o preço do tipo de aeronave solicitado.

Prototype

Basicamente este processo estabelece como deverá funcionar nosso projeto caso seja necessário clonar um objeto. Imagine que tenhamos que criar um sistema que represente uma "boiada". Legal, o que precisamos fazer primeiro é abstrair o que é um boi e quais são seus comportamentos, ótimo, o boi tem que andar, comer, respirar etc. Agora para que não tenhamos que criar uma classe chamada boi e outra chamada vaca, iremos criar uma classe chamada Bovino e esta terá uma classe Prototype que irá clonar a classe Bovino para podermos instanciar um objeto do tipo Vaca e outro do tipo Boi. Ambas com as mesmas características, mas com alguns detalhes específicos que as diferem.

Basta imaginar uma situação do mundo real, imagine que você tenha um animal que será clonado, o cientista que irá fazer esta façanha é nossa classe Prototype e o clone será nosso Concrect Object. Apesar de tanto o animal "original" quanto o animal clonado terem as mesmas células, com as mesmas estruturas biológicas não quer dizer que eles terão comportamentos exatamente idênticos. Tudo irá depender do habitat em que cada um for criado. Vamos agora imaginar a construção de um software. Por exemplo, o tipo primitivo String, ele pode ser aplicado ou instanciado em várias partes do software, mas cada instância poderá ter um comportamento diferente da instância anterior.

Singleton

Chegamos ao quinto e último padrão de criação. Agora que já entendemos por alto o que são e para que servem cada um destes padrões, chegamos ao foco principal do nosso artigo. Neste padrão além da explicação como nos anteriores, teremos o desenvolvimento de uma classe de conexão completa. Esta classe será bastante útil aos desenvolvedores PHP, pois ela centraliza as requisições ao banco de dados e não é necessário se preocupar com abrir e fechar a conexão com o banco a cada nova solicitação, pois a classe se preocupa com isto.

Pois bem, o que é, e para que serve o padrão Singleton? Este padrão pode ser considerado como padrão mais importante na criação de um software. Isto se justifica pelo seguinte, ele é utilizado quando você tem a necessidade de gerenciar algum objeto para que tenha a certeza, de que este objeto terá uma única instância e que possa ser utilizado em todo software, alguns exemplos poderiam ser citados como o pool de uma determinada impressora, o controle de janelas abertas no sistema ou o que é mais comumente utilizado, a conexão com o banco de dados.

Este padrão especifica que devemos instanciar o objeto da classe em questão apenas em um local, e que todas as classes que venham necessitar de um objeto desta, devem fazer a requisição a este único objeto instanciado. Esta classe geralmente possui um método estático para que este método crie a instância de objetos desta classe.

Nota

É comum desenvolvedores abrirem conexões com banco de dados e simplesmente esquecerem de fechá-las. Isto quando se tem um número pequeno de usuários consumindo o sistema não é um problema, mas imaginando que isso possa acontecer em um site, por exemplo, que tenha um número de visitas diárias em torno de 30.000 (trinta mil), este será um sério problema, pois o sistema gerenciador de banco de dados perderá parte de seu desempenho devido ao número excessivo de conexões abertas e possivelmente pode chegar ao ponto de travar, sendo necessária sua reinicialização. Com a aplicação deste padrão na criação este problema estará resolvido.

Na **Figura 2** teremos uma demonstração gráfica da classe a ser implementada para conexão com banco de dados. Nosso foco aqui não é mostrar como se desenvolver corretamente a orientação objetos, mas mostrar como se utilizar corretamente os padrões, então caso for implementar a classe em seu projeto é recomendável que implemente todos os métodos get e set de acordo com os atributos especificados na classe.

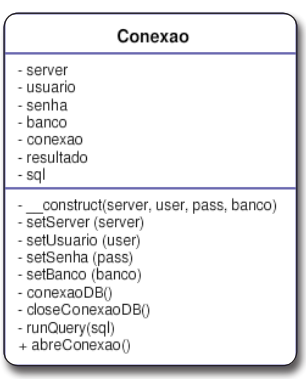


Figura 2. Classe de conexão

Note que todos os métodos desta classe são privados, exceto o método abreConexao() que é público e estático. Será através deste método que faremos a comunicação com nossa base de dados. Agora vamos iniciar o desenvolvimento da classe em questão. Na **Listagem 2** está descrito o cabeçalho da classe com os atributos pré-definidos e o método construtor. Vamos aproveitar a implementação desta classe para que além de apenas exemplificar como se aplica o padrão Singleton, mostrar como se documenta corretamente uma classe PHP. Com a documentação

adequada fica muito mais fácil o entendimento e a legibilidade do código. Ao final da implementação desta classe será exemplificado como consumir seus métodos sem ter a necessidade de criar uma nova instância em cada método que for necessário executar algum comando SQL. Esta instância será criada na própria classe de conexão que estamos implementando.

Listagem 2. Primeira parte da classe de conexão

```
01 <?php
02 /* Classe responsável pela conexão com o banco */
03   /**
04   * @author Classe desenvolvida por Dheyson Wildny (dheysonws@gmail.com)
05   * @category Conexão com Base de Dados
06   */
07    class Conexao {
08
09       /* Declaração dos atributos da classe */
10       private $servidor;
11       private $usuario;
12       private $senha;
13       private $banco;
14       private $conexao;
15       private $resultado;
16       private $sql;
17
18       /**
19        * @param String $server
20        * @param String $user
21        * @param String $pass
22        * @param String $banco
23        */
24       private function __construct($server, $user, $pass, $banco) {
25             $this->setServidor($server);
26             $this->setUsuario($user);
27             $this->setSenha($pass);
28             $this->setBanco($banco);
29       }
```

A Linha 7 é a declaração da classe que será implementada. Entre as Linhas 10 até a Linha 16 temos a declaração dos atributos da classe. Na Linha 24 temos o início da implementação do método construtor da classe, diferentemente de outras linguagens como Java por exemplo, no PHP a declaração do método construtor ao invés de usar o próprio nome da classe aconselha-se usar a palavra reservada __construct(), mas não quer dizer que se criarmos o método construtor com o nome da classe esteja feito de forma errada, apenas vamos seguir as recomendações do PHP. Este método apenas alimenta os atributos da classe com as informações necessárias para realizar a conexão com o banco de dados.

Na **Listagem 3** veremos a segunda parte da classe de conexão. Entre as Linhas 02 à Linha 13 temos somente os métodos set da classe que são responsáveis por alimentarem os atributos da classe. Na Linha 16 iniciamos os métodos que realmente dão sentido a esta nossa classe. Este é o método nominado como ConexaoBD, responsável por abrir a conexão com o banco e selecionar a base de dados necessária. Na Linha 17 estabelecemos que o atributo da classe chamado "conexao" receberá o resultado da função que faz a conexão com a base de dados MySQL - mysql_connect. Esta por sua vez nos retorna um valor booleano, então na Linha 18 verificamos se o retorno desta função foi falso ou verdadeiro. Caso a conexão tenha sido feita com sucesso então é verificado se é possível conectar a base de dados informada no servidor em que foi estabelecida a conexão, caso a seleção da base de dados não tenha sido feita com sucesso, então é devolvida a mensagem ao usuário final e finaliza a aplicação, senão não é necessário executar mais nada pois a conexão com o banco já foi feita e a base de dados já está pronta para uso.

Listagem 3. Segunda parte da classe de conexão

```
01       /* Sets */
02       private function setServidor($server) {
03             $this->servidor = $server;
04       }
05       private function setUsuario($user) {
06             $this->usuario = $user;
07       }
08       private function setSenha($pass) {
09             $this->senha = $pass;
10       }
11       private function setBanco($banco) {
12             $this->banco = $banco;
13       }
14
15       /* Método que abre a conexão com o Banco de Dados */
16       private function ConexaoDB() {
17             $this->Conexao = mysql_connect($this->servidor,
         $this->usuario, $this->senha);
18             if(!$this->Conexao) {
19                   echo("<p>N&atilde;o foi poss&iacute;vel conectar-se
                           ao servidor MySQL.
20                   <b>Erro MySQL: ".mysql_error()."</p>");
21                   exit();
22             } elseif (!mysql_select_db($this->banco, $this->Conexao)) {
23                   echo("<p>N&atilde;o foi poss&iacute;vel selecionar o
                           Banco de dados desejado.
24                   <br>Erro MySQL: ".mysql_error()."</p>");
25                   exit();
26             }
27       }
```

Agora o próximo passo é criar o método para fechar a conexão e criar um método para executar rotinas na base de dados, conforme visto na **Listagem 4**.

Como podemos ver na **Listagem 4**, temos a apresentação dos dois métodos seguintes. O método closeConexaoDB fica responsável apenas por fechar a conexão com a base de dados. O método runQuery, é responsável por executar uma determinada rotina dentro da base de dados, com este método poderemos executar todas as nossas operações CRUD, na Linha 08 é chamado o método ConexaoDB do objeto instanciado, ou seja, dizemos ao objeto instanciado que é hora de abrir a conexão no servidor e preparar a base de dados desejada para podermos trabalhar. Na Linha 10 é executada a função dentro do banco de dados com o comando que foi recebido através do parâmetro, o resultado desta execução é armazenado no atributo resultado, e caso tenha sido devolvido algum valor nesta rotina então a conexão com o banco é fechada e o método runQuery devolve este valor, senão a conexão é fechada e a aplicação é finalizada. Note que neste caso fizemos de forma diferente do método ConexaoDB, que primeiro devolvemos a mensagem através da função echo e só depois chamamos a função exit para finalizar a aplicação, isto foi feito para mostrar que podemos passar a mensagem que será retornada ao usuário já na função exit sem a necessidade do echo. Agora veremos o último método desta classe na **Listagem 5**.

Listagem 4. Terceira parte da classe de conexão

```
01      /* Método que fecha a conexão com o banco de dados */
02      private function closeConexaoDB() {
03          return mysql_close($this->Conexao);
04      }
05
06      /* Método que executa comando SQL */
07      private function runQuery($sql) {
08          $this->ConexaoDB();
09          $this->sql = $sql;
10          $this->resultado = mysql_query($this->sql);
11          if($this->resultado) {
12              $this->resultado;
13              return $this->resultado;
14          } else {
15              $this->closeConexaoDB();
16              exit("<p>N&atilde;o foi poss&iacute;vel executar o
17                  comando solicitado.
18                  <br>Erro MySQL: ".mysql_error()."</p>");
19          }
20      }
```

Listagem 5. Quarta parte da classe de conexão

```
01      /* Método que instancia um objeto da classe de conexão */
02      static public function abreConexao() {
03          $con = new Conexao('localhost','root','','dheysonws');
04          return $con;
05      }
06  }
07
08  ?>
```

Este último método representado na **Listagem 5** não possui nada muito complexo, um dos destaques para este método é a sua declaração. Este além de ser o único método público da classe, é também estático. E é nele que instanciamos o objeto de conexão, ou seja, somente esta classe que fará instâncias de objeto de conexão, centralizando assim o nosso acesso.

Ótimo, agora que já entendemos como implementar uma classe seguindo o padrão Singleton, iremos aprender como utilizar esta classe de maneira correta. Imagine que temos a seguinte situação: temos que desenvolver um projeto que possua uma tabela de contatos, então neste caso teremos uma classe de Conexao e uma classe que seria chamada de Contato. Nesta classe teremos um método para incluir um novo registro no banco de dados, então para conseguir a conexão com o banco e executar esta rotina utilizaremos a classe que acabamos de implementar.

A **Listagem 6** faz uma demonstração básica de como seria esta classe de Contatos, nela temos dois atributos: "id" e "nome", um método construtor e o método para inclusão de contatos no banco de dados. Nada de diferente do que estamos acostumados, exceto pela forma com que é chamado o método runQuery da classe de conexão. Na Linha 20 a variável resultado recebe o retorno da instrução SQL de inserção, quando escrevemos o código: "Conexao::abreConexao" estamos dizendo ao PHP que queremos utilizar um método estático da classe de conexão, chamado abreConexao. Ao fazermos referência a este método, automaticamente estamos criando uma instância de um objeto Conexao através de sua classe e então já podemos utilizar seus métodos, no caso o runQuery que é um método privado e só pode ser acessado dentro da própria classe. Após executar o método é verificado se foi ou não retornado algum valor do comando executado, se sim, o método retorna verdadeiro, caso contrário retorna falso.

Listagem 6. Classe de contatos

```
01  <?php
02      /* Classe responsável pelo tratamento dos contatos */
03      /**
04       * @author Classe desenvolvida por Dheyson Wildny (dheysonws@gmail.com)
05       * @category Manipulação de Contatos
06       */
07      class Contato {
08
09      /* Atributos da Classe */
10      private $id;
11      private $nome;
12
13      /* Método Construtor */
14      public function __construct() {
15
16      }
17
18      /* Método Construtor */
19      public function incluir() {
20          $resultado = Conexao::abreConexao()->runQuery("INSERT INTO
            (id, nome)
21          VALUES ('".$this->$id."', '".$this->nome."') ");
22          if ($resultado) {
23              return true;
24          } else {
25              return false;
26          }
27      }
28      }
29
30  ?>
```

Conclusão

Com isso finalizamos nosso artigo. O foco principal deste artigo foi abordar o padrão Singleton que é o que deve ser sempre utilizado quando temos que desenvolver um projeto orientado a objetos. Utilizo esta classe de conexão há bastante tempo em 100% dos projetos em que trabalho e isso facilita muito o acesso ao banco de dados. Agora basta analisar melhor seus projetos e descobrir se você precisa ou não utilizar algum destes padrões, projetar seu sistema corretamente vai garantir um resultado melhor tanto para quem projeta quanto para quem desenvolve. Até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Cursos Online

Assinatura .net magazine PLUS

Mais conteúdo .NET por muito menos!

A revista **.net Magazine** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade .
Conheça abaixo os cursos já disponíveis.

Curso em destaque

Crie uma loja virtual completa

Confira neste curso Online como construir uma loja virtual completa no Visual studio 2005. Aprenda como criar o banco de dados, carrinho de compras, profile, página de erros e muito mais .

Confira o plano de aula completo:
www.devmedia.com.br/lojavirtual

A sua melhor opção de aprendizagem!
Assine a **.net Magazine** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Construindo relatórios com Crystal Reports e Visual Studio 2005
- Criando uma aplicação Web Completa
- Criando uma aplicação client/server no Visual Studio 2005
- Aprenda a criar um blog com ASP.NET
- Curso de C#
  * Curso em andamento

DevMedia group Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.